

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

CONCEPTS, APPLICATIONS AND ANALYSIS OF A SUBMARINE BASED WIRELESS NETWORK

by

William G. Wilkins Jr.

June 2001

Thesis Advisor
Second Reader

Xiaoping Yun
C.Thomas Wu

20011128 022

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE: Title (Mix case letters) Concepts, Applications and Analysis of a Submarine Based Wireless Network		5. FUNDING NUMBERS		
6. AUTHOR(S)		8. PERFORMING ORGANIZATION REPORT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE		
13. ABSTRACT (maximum 200 words) As Information Technology tools continue to improve, we must take advantage of this wave by developing wise solutions to help automate many daily tasks presented onboard submarines. Java based applications and Commercial-Off-The-Shelf (COTS) technology provides us low cost solutions that increase the availability and mobility of the information we seek. Small pen based computers and wireless LANS allow us to create dynamic and distributable applications that can route paperwork or fight casualties. It is imperative we take full advantage of these technologies in the design of our new submarines as well as in retrofit of our older ones. This thesis attempts to solve a key task, Damage Control (DC) communications, by designing a Java based application known as SWIPNet (Submarine Wireless Prototyped Network). This virtual grease board application uses multicast sockets to send standard DC and crew reports to all wireless handhelds that participate in a casualty. A proposed Virginia class wireless network, known as Non Tactical Data Processing System (NTDPS), was then analyzed to determine network efficiency in the presence of a SWIPNet and 14 other submarine type network loads. Demonstrations have proven that SWIPNet provides a more efficient way to communicate and can function effectively on the NTDPS.				
14. SUBJECT TERMS Wireless Local Area Network, mobile computing, Java, pen-based computing, PDAs, Handheld Computers, database, OPNET Modeler, Microsoft Access, Damage Control, Multicast Sockets			15. NUMBER OF PAGES 248	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**CONCEPTS, APPLICATIONS AND ANALYSIS OF A SUBMARINE BASED
WIRELESS NETWORK**

William G. Wilkins Jr.
Lieutenant, United States Navy
B.S., Auburn University, 1994

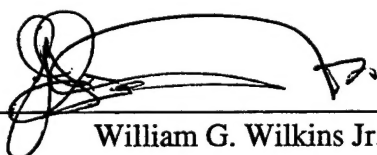
Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE
IN
COMPUTER SCIENCE**

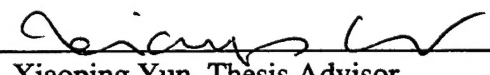
from the

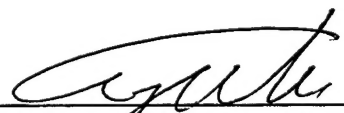
**NAVAL POSTGRADUATE SCHOOL
June 2001**


Author:


William G. Wilkins Jr.

Approved by:


Xiaoping Yun, Thesis Advisor


C. Thomas Wu, Second Reader


Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

As Information Technology tools continue to improve, we must take advantage of this wave by developing wise solutions to help automate many daily tasks presented onboard submarines. Java based applications and Commercial-Off-The-Shelf (COTS) technology provides us low cost solutions that increase the availability and mobility of the information we seek. Small pen based computers and wireless LANS allow us to create dynamic and distributable applications that can route paperwork or fight casualties. It is imperative we take full advantage of these technologies in the design of our new submarines as well as in retrofit of our older ones.

This thesis attempts to solve a key task, Damage Control (DC) communications, by designing a Java based application known as SWIPNet (Submarine Wireless Prototyped Network). This virtual grease board application uses multicast sockets to send standard DC and crew reports to all wireless handhelds that participate in a casualty. A proposed Virginia class wireless network, known a Non Tactical Data Processing System (NTDPS), was then analyzed to determine network efficiency in the presence of a SWIPNet and 14 other submarine type network loads. Demonstrations have proven that SWIPNet provides a more efficient way to communicate and can function effectively on the NTDPS.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	GOALS FOR THIS THESIS	1
B.	THESIS OUTLINE	1
II.	DETAILED SUMMARY OF WIRELESS GROUP RESEARCH	3
A.	SCOPE	4
B.	THEORY ^{1,4,6}	5
1.	Channel Path Characteristics ^{4,6}	5
2.	Spectrum Types ^{4,6}	7
a.	<i>FH/SS</i>	7
b.	<i>DS/SS</i>	8
c.	<i>Comparison between FH/SS and DS/SS</i>	8
3.	Electromagnetic Interference (EMI) ¹	8
C.	STANDARDS ^{2,4,5,6}	9
1.	OSI Model ^{4,5,6}	9
2.	CSMA/CD ^{4,6}	10
3.	CSMA/CA ^{4,5,6}	11
a.	<i>Hidden Node</i>	15
b.	<i>Fading Effects</i>	16
4.	Security ²	16
D.	HARDWARE ^{ALL}	17
1.	Handheld Devices ^{ALL}	18
2.	Wearable Devices ^{2,3,4,5}	19
3.	Performance of Above Tested COTS Equipment ^{1,4,5}	19
4.	Wireless Components ^{ALL}	20
E.	SOFTWARE DEVELOPMENT ^{3,5}	21
1.	Coding Languages ^{3,5}	21
2.	Feasibility Application ^{3,5}	22
F.	TESTING ^{ALL}	23
1.	Platform Environments ^{ALL}	23
a.	<i>USS Ohio Test (August/1997)</i> ¹	23
b.	<i>USS Harry S. Truman Hangerbay (March/1999)</i> ^{3,4,5}	23
c.	<i>USS Memphis (August/1999)</i> ^{4,6}	25
d.	<i>NPS Labs (3/2000)</i> ^{2,4,6}	25
G.	SYSTEM REQUIREMENTS ¹	27
1.	Requirements ¹	27
H.	CHAPTER SUMMARY	27
III.	APPLICATION AND NETWORK REQUIREMENTS	29
A.	TCP/IP MODEL, THE WORKING MODEL, [HUGHES97]	29
B.	INTERNET PROTOCOL (IP) ADDRESSING	30
C.	THE JAVA LANGUAGE	31

	1.	Why Java?.....	31
	2.	Security Issues	31
D.		SOCKET BASED COMMUNICATION	32
	1.	What is a Socket?.....	32
	2.	How Sockets Can Be Used for Communication.....	32
	3.	Submarine Applications and Socket Based Communication	32
	a.	<i>Drill Control</i>	32
	b.	<i>Supply System</i>	32
	c.	<i>Log Taking</i>	33
	d.	<i>Preventive Maintenance (PM)</i>	33
	e.	<i>Sub to Sub Data Transfers</i>	33
	f.	<i>Casualty/Damage Control</i>	33
	g.	<i>Standard Operations</i>	34
	h.	<i>Troubleshooting</i>	34
	4.	Specific Socket Communications [Mahmoud00]	34
	a.	<i>Transmission Control Protocol (TCP) Sockets</i>	34
	b.	<i>User Datagram Protocol (UDP) Sockets</i>	35
	c.	<i>Broadcast Sockets</i>	35
	d.	<i>Multicast Sockets</i>	35
	e.	<i>Remote Method Invocation (RMI)</i>	35
	f.	<i>Common Object Request Broker (CORBA)</i>	36
E.		THE DC COMMUNICATION MODEL	36
F.		BUILDING A DAMAGE CONTROL APPLICATION.....	39
	1.	General Software Constraints	39
	a.	<i>Scalability</i>	39
	b.	<i>Mobility</i>	39
	c.	<i>Multithreaded</i>	39
	d.	<i>Operate Similar to Current DC Communications</i>	39
	e.	<i>Reliability Higher Than Current DC Communications</i>	39
	f.	<i>Easy to Setup and Use</i>	40
	g.	<i>Network Capable</i>	40
	h.	<i>Fast Data Transfer</i>	40
	i.	<i>Customizable Configuration to the Ships Needs</i>	40
	j.	<i>Applet and/or Application Capable</i>	40
	k.	<i>Persistence Protection</i>	41
	2.	General Hardware Constraints.....	41
	a.	<i>Open Operating System</i>	41
	b.	<i>Network Connectivity</i>	41
	c.	<i>Rugged</i>	41
	d.	<i>Long Batter Life</i>	41
	e.	<i>Mobile Input Method</i>	41
	f.	<i>Mobile View Method</i>	42
	g.	<i>Comfort</i>	42
	h.	<i>Storage</i>	42
G.		CHAPTER SUMMARY	42

IV.	SUBMARINE DC IMPLEMENTATION IN JAVA.....	43
A.	GOALS	43
B.	SCOPE OF THE PRODUCT	43
C.	FEATURES OF SWIPNET	44
1.	Desired Features	44
a.	<i>Standard DC Reports</i>	44
b.	<i>Voice and Video</i>	44
c.	<i>Persistent Storage</i>	44
d.	<i>User Interface</i>	44
e.	<i>Availability</i>	44
f.	<i>Security</i>	44
g.	<i>Encryption</i>	44
h.	<i>Maintainability</i>	45
i.	<i>Customization</i>	45
j.	<i>Post Drill Feedback</i>	45
2.	Key Design Features.....	45
a.	<i>Sender/Listener Approach</i>	45
b.	<i>Multicast</i>	45
c.	<i>Applet/Application</i>	46
D.	OLD VS NEW COMPARISON.....	46
1.	Application is Better Than Applet.....	46
2.	No Bottlenecks.....	47
3.	Less Error Prone	47
4.	No Single Point of Failure	47
5.	Easier to Deploy to a Ship	47
6.	More Scalable	48
E.	SYSTEM SPECIFICATIONS	48
1.	Package Layout.....	48
2.	Class Summary	50
3.	Key Method Summary.....	52
F.	GRAPHICAL USER INTERFACE (GUI) LAYOUT	54
1.	In General	54
a.	<i>Color Use</i>	54
b.	<i>Compactness</i>	55
c.	<i>GUI Components</i>	55
2.	Launching (DCNet object) Console	55
3.	Sending (Client object) Console.....	56
4.	Listening GUI (Server object) Console	57
5.	Deploying the Application for Use.....	58
G.	CHAPTER SUMMARY.....	58
V.	SOLVING THE PERSISTENT DATA PROBLEM.....	61
A.	STORAGE MODELS.....	61
1.	Flat Files	61
2.	Relational Databases.....	61
3.	Object Oriented Databases	62

	4.	eXtensible Markup Language (XML)	62
B.		CONNECTION MODELS	63
	1.	Java Database Connectivity (JDBC)	63
	2.	Server Side Models.....	63
C.		DC DESIGN REQUIREMENTS	63
	1.	Initialization.....	64
	2.	Power Outage Recovery	64
D.		PERSISTENCE DESIGN USING MICROSOFT ACCESS	64
	1.	<i>Entity</i> Description	65
	2.	Relationships.....	67
	3.	Microsoft Access Implementation.....	68
	4.	Connecting the Database	69
E.		DATA INTEGRITY FEATURES USING MICROSOFT ACCESS.....	69
	1.	Security.....	69
	a.	<i>Encryption</i>	69
	b.	<i>User and Group Accounts</i>	69
	c.	<i>DB Password</i>	70
	d.	<i>Security Wizard</i>	70
	2.	Protective Locking of Data in a Multiple User Environment.....	70
	a.	<i>Record Locking</i>	70
	b.	<i>Page Locking</i>	70
	c.	<i>Table and Recordset Locking</i>	70
	d.	<i>Opening an Entire Database with Exclusive Access</i>	70
	3.	Data Integrity	71
	4.	DB Recovery and Backup	71
F.		CHAPTER SUMMARY.....	72
VI.		SUBMARINE NETWORK ANALYSIS	73
A.		GOALS	73
B.		DEVELOPMENT PLAN.....	74
C.		MODIFYING THE ORIGINAL DESIGN.....	75
D.		ASSUMPTIONS OF MODIFIED LAN:.....	76
	1.	LAN Classification	76
	2.	Link Classification.....	77
	3.	Fiber Interface Boxes (FIB) to Switch Conversion	77
	4.	Grouping of Traffic into Subnets.....	77
	5.	Placement of Components	78
	6.	Selection of Backbone Switches.....	78
	7.	Others.....	78
E.		LAYOUT OF THE SUBMARINE NETWORK.....	79
	1.	Physical Layout	79
	a.	<i>Submarine Cross Section Description:</i>	79
	b.	<i>Control Center and Link Layout Description</i>	80
	c.	<i>Submarine LAN Component Description</i>	81
F.		KEY SERVICES AND MEASURED VALUES USED IN STUDY	82
	1.	Key Services	82

	a.	<i>Class Roaming Service</i>	82
	b.	<i>Application Service</i>	83
	c.	<i>Plug-in and Printing Service</i>	83
2.		Metrics Studied during Simulation	83
	a.	<i>Packet Based Metrics (TCP/IP Network layer)</i>	83
	b.	<i>Application Metrics (TCP/IP Application layer)</i>	84
3.		Parameters Defined for Simulation	84
	a.	<i>System Parameters</i>	84
	b.	<i>Workload Parameters</i>	85
4.		Factors Varied during Simulation	85
G.		OPNET MODEL	85
	1.	OPNET Model Assumptions	86
	2.	Problems Encountered	86
	3.	OPNET Diagram	87
	4.	Configuration of the Simulation Components	88
	a.	<i>Wireless Client (Example of HTTP client)</i>	88
	b.	<i>Access Point (Example of AccessPoint4)</i>	89
	c.	<i>Domain Controller (Primary Domain Controller)</i>	89
	d.	<i>Workload Profile (All profiles developed)</i>	90
	e.	<i>Application (Example of HTTP6KB)</i>	91
	5.	Other Components	92
H.		WORKLOADS AND TASKS DESIGN	92
	1.	Submarine DC Drills	93
	2.	Log Taking	93
	3.	Repair Maintenance Management	93
	4.	Supply Inventory	93
	5.	Preventative Maintenance	93
	6.	Message Routing	93
	7.	Watch Bill Scheduling	93
	8.	Refit Planning	93
	9.	Plan of the Day (POD) Deployment	94
	10.	Qualification System	94
	11.	Fitness Reports (FITREP) and Evaluations (EVAL) Tracking	94
	12.	General Record Storage and Retrieval	94
	13.	Online Training	94
	14.	Online Ship Inspection and Exam History	94
	15.	Crew Leisure Activities	94
I.		WORKLOAD MATRIX AND TIMEFRAMES	95
	1.	Matrix	95
	2.	Timeframes	96
	3.	Workloads Assumptions	96
	a.	<i>Mean Application Frequency for Given Timeframe</i>	96
	b.	<i>Mean Size of Application Loading</i>	96
	c.	<i>Timeframe#</i>	98
	d.	<i>Task Areas</i>	98

	<i>e. Servers</i>	<i>98</i>
	<i>f. Application Services Selection.....</i>	<i>98</i>
4.	Workloads Plotted	98
5.	Workload Grouping	99
6.	Application Grouping	100
7.	OPNET Simulation Results	100
	<i>a. Subnet-to-Subnet Throughput (bps).....</i>	<i>100</i>
	<i>b. Application Mean Response Time</i>	<i>101</i>
	<i>c. Application Load (Wireless Clients).....</i>	<i>102</i>
	<i>d. Ethernet Delay</i>	<i>103</i>
J.	ANALYTICAL MODELING	103
	1. Definition of Problem	103
	2. Assumptions of the Analytical Model	104
	3. Approach.....	105
	4. Analytical Results.....	106
K.	SENSITIVITY ANALYSIS	107
	1. Workloads Applied.....	107
L.	CHAPTER SUMMARY	110
VII.	RECOMMENDATIONS AND CONCLUSION.....	113
	A. SUMMARY	113
	B. RECOMMENDATIONS FOR FUTURE WORK.....	114
	C. FINAL CONCLUSION.....	115
	APPENDIX A – SELECTED SOURCE CODE.....	117
	LIST OF REFERENCES.....	205
	INITIAL DISTRIBUTION LIST.....	209

LIST OF FIGURES

Figure 1 - Channel Characteristics	6
Figure 2 - Types of Fading	7
Figure 3 - OSI Model.....	10
Figure 4 - CSMA/CD Network.....	11
Figure 5 - CSMA/CA (Infrastructure Mode) [From: Kurose01].....	12
Figure 6 - CSMA/CA (ADHOC Mode) [From: Kurose01]	12
Figure 7 - Collision Avoidance (Basic Access Mechanism) [From: Kurose01]	13
Figure 8 - Collision Avoidance (RTS/CTS) [From: Kurose01]	14
Figure 9 - Hidden Node (a) and Fading Effects (b) [From: Kurose01]	15
Figure 10 - Handheld and Wearable Devices That Were Evaluated	19
Figure 11 - Evaluation of Selected COTS Equipment.....	20
Figure 12 - Wireless Components That Were Evaluated	21
Figure 13 - Java Virtual Machine Flow Path.....	22
Figure 14 - USS Harry S. Truman Hangerbay	24
Figure 15 - Average Throughput vs. Number Clients (USS Truman)	24
Figure 16 - Access Point and Throughput Coverage (USS Memphis).....	25
Figure 17 - Lucent WaveLAN (Turbo) Line Of Sight	26
Figure 18 - System Cost Wireless (Subsystem)	27
Figure 19 - TCP/IP Model	30
Figure 20 - Damage Control Status Board - USS Batfish	37
Figure 21 - Package Level Diagram	50
Figure 22 - SWIPNet Class and Method Level Diagram	52
Figure 23 - Launching (DCNet object) Console.....	56
Figure 24 - Sending (Client object) Console	56
Figure 25 - Listening GUI (Server object) Console	57
Figure 26 - Entity Relationship (ER) Diagram.....	66
Figure 27 - Relationship Schema/Relationships and Attributes	68
Figure 28 - Flowchart of Network Design.....	74
Figure 29 - "Original Design" LAN	75
Figure 30 - "Modified" LAN.....	76
Figure 31 - "Modified" LAN Physical Layout	79
Figure 32 - Control Center Layout	81
Figure 33 - OPNET Model (with expanded Gold Subnet)	87
Figure 34 - Example Wireless Client Configuration	88
Figure 35 - Example Access Point Configuration	89
Figure 36 - Example Domain Controller Configuration.....	90
Figure 37 - Profile Configuration	91
Figure 38 - Example Application Configuration	91
Figure 39 - Other Modeled Components	92
Figure 40 - Workload Selection Flow Chart.....	92

Figure 41 - Workload Matrix.....	95
Figure 42 - Timeframes Defined	96
Figure 43 - Twenty-Four Hour Loading Results	99
Figure 44 - Subnet-to-Subnet Throughput [bps vs. minutes (m)]	101
Figure 45 - a) Application Response Time b) Application Response Time.....	102
Figure 46 - Application Load From Wireless Clients [bps vs. minutes (m)]	102
Figure 47 - Ethernet Delay for a Basic Workload	103
Figure 48 - Longest Path scenario	104
Figure 49 - Calculation of Mean Ethernet Delay E(D).....	105
Figure 50 - Delay vs. Arrival Rate.....	107
Figure 51 - FTP Response Time (sec vs. minutes) [All Simulation Runs]	109
Figure 52 - Ethernet Delay E(D) (sec vs. minutes) [All Simulation Runs].....	109
Figure 53 - Response Times a) x5 Workload b) x38 with No Video	110

LIST OF TABLES

Table 1 - IP Classes and Address Range	31
Table 2 - SWIPNet Package Summary.....	49
Table 3 - SWIPNet Class Summary	52
Table 4 - Key SWIPNet Method Summary.....	54
Table 5 - Inter-Arrival Times	100
Table 6 - Longest Path E(D) Comparison Table	106

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the financial support of NAVSEA PMS450 in sponsoring this research.

The author would also like to thank Professors Xiaoping Yun and Thomas Wu for their guidance and support during this research effort. I would also like to thank the previous members of the wireless research group that provided me assistance and guidance in getting started with this research.

Finally, the author would like to thank his spouse, Emily, and his boys, Christopher and Michael, for their enduring patience and understanding during the writing of this research. Your love and humor provided the support I needed to bring this work to a close, thank-you.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS AND/OR ABBREVIATIONS

ACK	Acknowledgement
AP	Access Point
API	Applications Programming Interface
A/V	Audio Visual
AWGN	Additive White Gaussian Noise
BSS	Basic Service Set
BDC	Backup Domain Controllers
CORBA	Common Object Request Broker
COTS	Commercial Off the Shelf
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DB	Database
DBA	Database Administrator
DC	Damage Control
DCC	Damage Control Central
DCF	Distributed Coordination Function
DES	Digital Encryption Standard
DIFS	Distributed Inter Frame Space
DS/SS	Direct Sequence Spread Spectrum
DTD	Document Type Definition
EAB	Emergency Air Breathing
EMC	Electro Magnetic Compatibility
EMI	Electro Magnetic Interference
ER	Entity Relationship
EOOW	Engineer Officer Of the Watch
FH/SS	Frequency Hop Spread Spectrum
FIB	Fiber Interface Box
FSK	Frequency Shift Keying

FTP	File Transfer Protocol
GUI	Graphical User Interface
HIPERLAN	High Performance Radio Local Area Network
HTTP	Hypertext Transfer Protocol
Hz	Hertz
IP	Internet Protocol
JIT	Just in Time
JVM	Java Virtual Machine
J2EE	Java 2 Enterprise Edition
JSP	Java Server Page
JDBC	Java Database Connectivity
JRE	Java 2 Runtime Environment
LAN	Local Area Network
LOS	Line-Of-Sight
MAC	Medium Access Control
MIC	Man In Charge
ML	Markup Language
NAVSEA	Naval Sea System Command
NFTI	Navy Field Thermal Imager
NPS	Naval Postgraduate School
NSSN	New Attack Submarine
NTDPS	Non Tactical Data Processing System
OBA	Oxygen Breathing Apparatus
OOD	Officer of the Deck
ORSE	Operational Reactor Safety Exam
OS	Operating System
OSI	Open Systems Interconnection
PAN	Personnel Area Network
PB	Prewatch - Based
PPB	Paperwork/Postwatch - Based

PDC	Primary Domain Controller
PHY	Physical Layer
PIB	Pre-Inspection Based
PM	Preventive Maintenance
PMS	Preventive Maintenance System
QPSK	Quadrature Phase Shift Keying
RAM	Read Access Memory
RB	Random Based
RDB	Random Day Based
RIP	Routing Information Protocol
RMI	Remote Method Invocation
RTS/CTS	Request-To-Send/Clear-To-Send
SIFS	Short Inter Frame Space
SOF	Special Operations Force
SQL	Structured Query Language
SSL	Secure Socket Layer
SWIPNet	Submarine Wireless Prototyped Network
TCP	Transmission Control Protocol
TRE	Tactical Readiness Exam
UDP	User Datagram Protocol
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WB	Watch Based
WLAN	Wireless Local Area Network
XML	eXtensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The dawning of the information age has presented the submarine force with an interesting dilemma of how, when or even why should we use certain technological advances to increase efficiency of key processes onboard a submarine. Wireless networks, handheld wireless computers and distributed software packages are examples that have increased our potential to share and distribute information. Most of these examples are available as Commercial Off the Shelf (COTS) products. These products have the potential to help crewmembers become more efficient in managing task, coordinating workloads and even combating casualties that can occur onboard a submarine. This thesis is part of an outgoing research project that involves the testing, analysis and design of submarine applications deployed on a wireless network.

A. GOALS FOR THIS THESIS

The main thrust of this thesis is to investigate the real world technological options available from COTS based broadband and wireless technologies. It will also encompass taking submarine concepts and apply them to different areas, like wireless LAN's (Local Area Networks) and pen based computers, wireless sensors, and broadband submarine based application that solve everyday tasks. A desired end result within this thesis is to design and implement a virtual Damage Control (DC) status board that can be used within a wireless LAN system and to tests its feasibility within a simulated Virginia Class wireless network.

B. THESIS OUTLINE

The following describes the way the remaining chapters are broken down. Chapter II will include an overview summary of previous work. This chapter provides a starting point for continued research within this thesis. Chapter III summarizes researched concepts that are needed to design and build a submarine based virtual DC

system. Chapter IV explains the design of this Java based application. Chapter V describes the database that was designed to provide persistent storage for the Java based application designed in chapter IV. Chapter VI describe the building of the Virginia class wireless LAN simulation using OPNET Modeler 7.0B. This chapter will explore the loading capabilities of this network and try to discover new insights on how to build this network better. Also this simulation will be performed to ensure that such a network can support the virtual DC application designed in Chapter IV. Chapter VII presents conclusions and recommendations for further study.

II. DETAILED SUMMARY OF WIRELESS GROUP RESEARCH

Students from Naval Postgraduate School (NPS) initially began efforts to determine the feasibility of wireless components aboard ships and submarines. It began in 1996 with the Submarine Wireless LAN (SWLAN) project between NPS, NAVSEA PMS450 and Electric Boat. This project initially conducted a market assessment of current wireless technologies and looked at what shipboard applications that would benefit from the integration of a wireless subsystem aboard naval vessels. Since that time countless wireless products have been evaluated. These products range from wearable computers to access points and wireless PCMCIA cards. Also several client/server applications have been developed to promote a proof concept with wireless technologies, including damage control and log taking. A series of wireless test have been conducted aboard the USS Ohio (SSBN 726), USS Harry S. Truman (CVN 75), USS Memphis (SSN 691), and within NPS Labs to gather field data. This data has been used to evaluate claims made by manufacturers and to identify the best wireless components. It is the goal of the SWLAN project to provide guidance on the newest wireless technology in the areas of testing and design to NAVSEA PMS450 and other Naval organizations.

This chapter presents an overview of topics covered previously by other graduate students. It is intended to summarize key research aspects and provide a foundation for continuing work, including this thesis. Specifically, this chapter pulls some of the key ideas from six graduate level theses that have been conducted within the SWLAN. Some areas have been expounded upon in greater detail from supplemental resources, but those familiar with these works could skip to Chapter III for the original work of this thesis.

A. SCOPE

The scope of this chapter is to summarize reoccurring themes presented in previous research. The themes, summarized from past research, can be broken down into six key areas:

- ❑ Theory^{1,4,6}
- ❑ Standards^{2,4,5,6}
- ❑ Hardware^{All}
- ❑ Software Development^{3,5}
- ❑ Testing^{2,3,4,5,6}
- ❑ System Requirements^{1,3}

The following theses cover these areas to different degrees. Credit is given via superscripts of areas these authors covered and some direction is provided for a reader who desires more detailed information. It is not intended to write all the specifics that are written in these theses, instead to summarize the more current ideas, list some interesting insights these authors have discovered and add some discussion from other references when warranted. The following works and authors have contributed to the SWLAN research group and their works are summarized within this chapter.

- ❑ [Debus98] - "Feasibility Analysis for a Submarine Wireless Computer Network Using Commercial off the Shelf Components"
- ❑ [Roemhildt99] - "Analysis and Vulnerabilities of Spread Spectrum Wireless Local Area Networks on Surface and Sub-Surface Combatants"

^{All} – All six theses addresses topic, ¹ - [Debus98], ² - [Roemhildt99], ³ - [Rothenhaus99], ⁴ - [Matthews99], ⁵ - [Sayat99], ⁶ - [McConnell00]

- [Rothenhaus99] - "Distributed Software Applications in Java for Portable Processors Operating on a Wireless LAN"
- [Matthews99] - "Analysis of Radio Frequency Components for Shipboard Wireless Networks"
- [Sayat99] - "Damage Control and Log Taking Java Applications for Shipboard Wireless LANs"
- [McConnell00] - "Testing and Evaluation of Shipboard Wireless Network Components"

B. THEORY^{1,4,6}

Three key areas of wireless systems is the channel characteristics, spectrum type and signal propagation. It is important to understand these characteristics in order to explain performance differences or effects of wireless components operating in a submarine or shipboard environment.

1. Channel Path Characteristics^{4,6}

Certain assumptions are made within any communication system. This section describes some assumptions that are made when describing shipboard channel paths. Three types of channels Rayleigh, Ricean, and AWGN (Additive White Gaussian Noise) are possible as shown in Figure 1. They are characterized by their direct path and multipath components. AWGN is the most basic channel path where a transmitted direct path signal experiences "freespace loss". This loss is proportional to the square of the distance between server and client. The next two are characterized in terms of large scale and small scale fading. Large scale fading is loss due to obstructions. Small scale

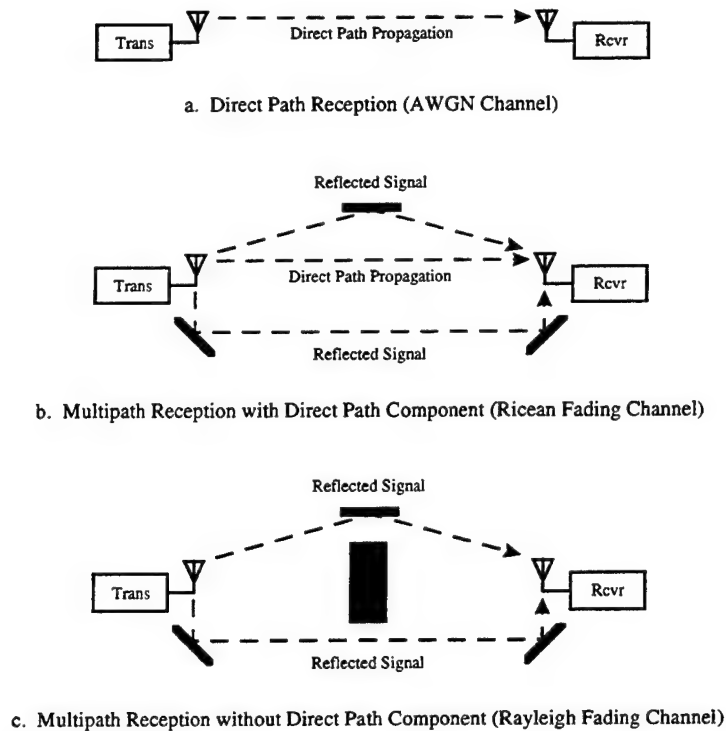


Figure 1 - Channel Characteristics

spreading is more complex and is broken down into time spreading and time variations. Time spreading is a signal phase that results from phase variations as a direct path and reflected signal that arrive with different phases and constructively and destructively combine to form the received signal. Time spreading results in a frequency select type fade (Figure 2), known as a Deep Fade, that has channel nulls. Time spreading can also result in flat fading which is more desirable since it fades the entire signal. Time variation-fading results due to relative motion between transmitter and receiver, which causes the propagation path to change. Time variation is further categorized as fast fade and slow fade, both of which are dependent on the velocity of the movement. Fast fade occurs when the symbol duration is greater than the coherence time and is similar to frequency select fading. Slow fade is when the coherence time is greater than the transmitted symbol duration. In a shipboard environment direct path components between the client and a base station (Access Point) rarely exist. Shipboard environment expected to exhibit slow, flat fading characteristics with occasional channel

nulls. So the channel that best characterizes a submarine environment is a Rayleigh channel. This concept is important when analyzing attenuation and coverage ability of different spectrums.

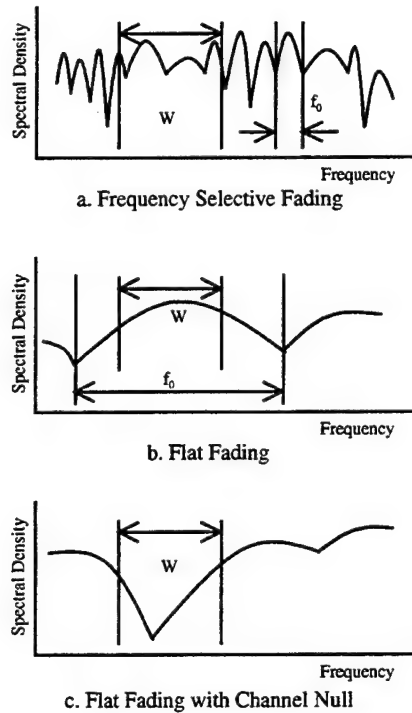


Figure 2 - Types of Fading

2. Spectrum Types^{4,6}

Spectrum types identify the method that wireless components use to propagate wireless signals through air. Two types are covered: Frequency Hop Spread Spectrum (FH/SS) and Direct Sequence Spread Spectrum (DS/SS). Both spectrum types are commercially available in today's wireless LAN products.

a. FH/SS

Frequency Hop passes a signal using a narrowband carrier that changes in frequency. The transmitter and receiver each know this pattern. In effect, FH/SS hops

from narrowband to narrowband within a wideband using each narrowband for a specific amount of time.

b. DS/SS

Direct Sequence uses a redundant bit pattern. This pattern is called the spreading or chipping code. Implementation is accomplished by modulating a narrowband signal with this chipping code. This makes DS/SS appear as a low power wideband noise to a narrowband receiver. In effect, you get a broadband signal by artificially increasing the modulation using the spreading code.

c. Comparison between FH/SS and DS/SS

LT Richard McConnell [McConnell00] provides an excellent comparison of the two spectrums in terms of coverage, reliability, immunity and scalability. He points out that DS/SS generally wins out in each category due to its more robust signal. This robustness is generally due to the fact that it spreads its signal across a wider spectrum. Therefore it is less affected by attenuation, as would a narrowband signal. Also the modulation technique, Quadrature Phase Shift Keying (QPSK) for DS/SS is more efficient than Frequency Shift Keying (FSK) used in FH/SS.

3. Electromagnetic Interference (EMI)¹

Electromagnetic interference (EMI) and electromagnetic compatibility (EMC) are two topics currently under research at NPS. The only prior research was within [Debus98] thesis. A summary is included for completeness within this chapter. More study is needed within this area and is currently being researched within another student's graduate thesis.

EMI is important because any equipment that comes onboard and transmits RF signals can potentially disrupt onboard systems. Two frequencies are normally encountered when dealing with wireless components. They are 900Mhz and 2.4 GHz, more recent industry focus on 2.4 GHz. Currently two documents cover the control of these transmitting devices. Those documents are MIL-STD-461D, *Requirements for the*

Control of Electromagnetic Interference Emissions and Susceptibility and OD-30303, *The HERO Design Guide*. Each document defines a maximum electric field constraint on all transmitting devices. This electric field value is expressed in V/m and equation (1) is used to convert this electric field to a power output value so it can be directly compared to any PCMCIA card. MIL-STD-461D sets a maximum electric field of 5V/m (10KHz to 40Ghz) and OD-30303 sets it at 13.5 V/m (900Mhz) if unity gain ($G_t = 1$) and a distance of 1 meter ($r=1$), are assumed. Then the corresponding maximum output is 833.3mW and 6.1W respectively. Most PCMCIA transmitting devices transmit at a maximum of 100mW and thus meet these requirements (at 1 meter). Also note that OD-30303 sets the maximum based on 900Mhz. Since higher frequencies only improve these limits, then the 2.4 GHz stations would also easily meet this requirement at one meter.

$$P_{out} = \frac{r^2 \times E^2}{30 \times G_t} \quad (1)$$

More research is needed to first challenge these limits and also explore hidden EMI effects that may not be prevented just by setting limits on max power output.

C. STANDARDS^{2,4,5,6}

1. OSI Model^{4,5,6}

The OSI model (Figure 3) provides a layered approach description that is used when talking about network systems. Although not actually implemented, it does provide a direct mapping to the implemented and very popular TCP/IP model. The TCP/IP model uses the Application, Transport (TCP), Network (IP), Data Link, and Physical layers that are described within the OSI model (see Chapter III for more detail on the TCP/IP model).

Application		
Presentation		
Session		
Transport		
Network		
Data Link	802.2 LLC	
Physical	802.3 Ethernet CSMA/CD	802.11 Wireless CSMA/CA

Figure 3 - OSI Model

The layers that are focused on within past research and this thesis reside in the Data Link (specifically the MAC layer) and the Physical Layer. These are where two types of networks, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), operationally reside and where they distinguish themselves. Layers above the Data Link layer are essentially the same in most networks and are not discussed here.

2. CSMA/CD^{4,6}

IEEE 802.3, CSMA/CD and Ethernet are all interchangeable words that describe a protocol that works within a CSMA/CD Network (Figure 4). This protocol deals with the lower two levels in the OSI model (the Physical and MAC sub layers). Common Physical Layer components are 100BaseT, 10BaseT, 100BaseFX, etc. On the MAC sub layer, when a network client has data to transmit, it first listens to the channel. If the idle (i.e. no transmitting clients) then the client transmits. If not idle, then the client waits for idle channel. When two clients transmit overlapping signals, a collision occurs. During

this “collision detection” process a jamming signal is issued and a random back off time is applied until another transmission is attempted. This process is repeated. Below is a typical bus connected CSMA/CD network that shows several wired clients that compete in the same collision domain. A bridge is also shown that connects this bus network to other networks (or LANs) and works inside the Data Link layer.

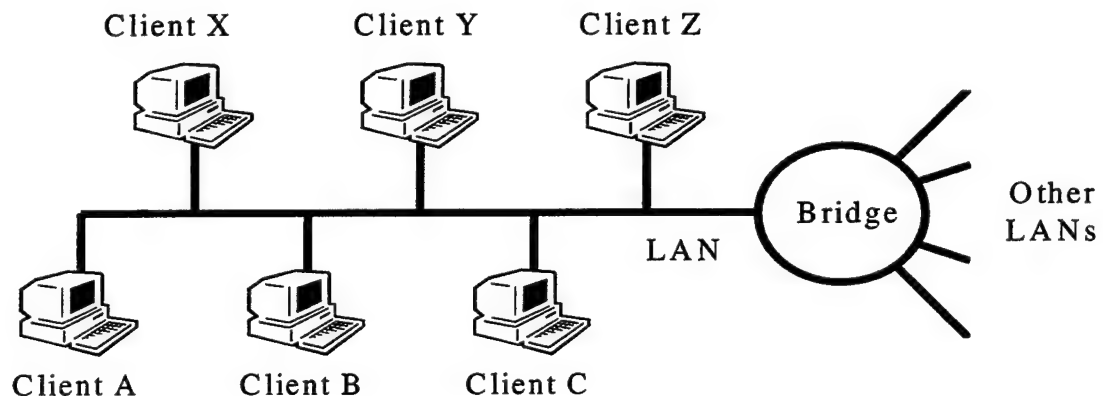


Figure 4 - CSMA/CD Network

3. CSMA/CA^{4,5,6}

IEEE 802.11, CSMA/CA and wireless LAN are also interchangeable. The fundamental building block of a wireless network is a basic service set (BSS), shown in Figure 5. This set consists of a base station, commonly referred to as an Access Point (AP) and one or more wireless stations (clients). These clients can be mobile or fixed. A BSS operates in what is called infrastructure mode [Kurose01], also shown in Figure 5. In this mode the Access Point provides an interface to the hardwired LAN and allows clients to access servers and other hardwired components.

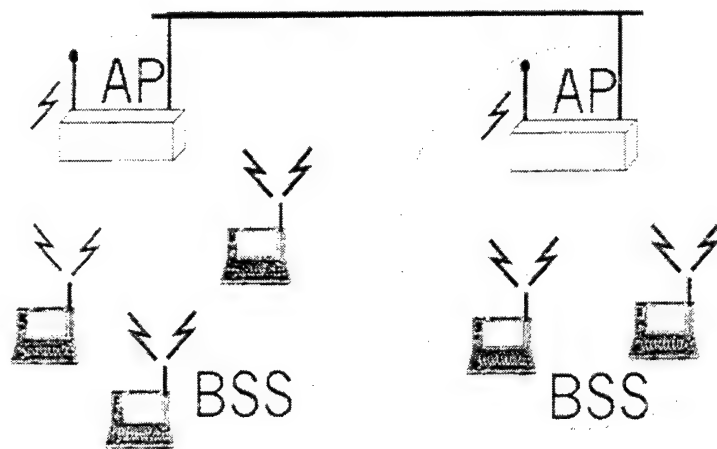


Figure 5 - CSMA/CA (Infrastructure Mode) [From: Kurose01]

Another mode, considered “on the fly” is called ADHOC Mode [Kurose01], shown in Figure 6. ADHOC modes operate without an Access Point. Here wireless clients connect directly with each other. The mode primarily investigated in this research is Infrastructure Mode, since ultimately the implementation is geared towards wireless network integration into a wired CSMA/CD network on board a submarine. However, it is noted that both modes function similarly on the Datalink layer.

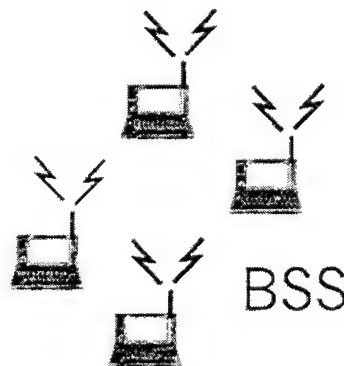


Figure 6 - CSMA/CA (ADHOC Mode) [From: Kurose01]

Datalink Operational Description [Bianchi00] - Wireless LANS do not use Collision Detection like CSMA/CD, because the ability to receive and transmit at the same time is not possible in a wireless environment.

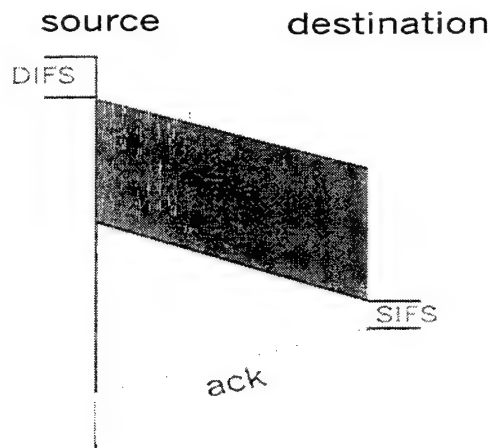


Figure 7 - Collision Avoidance (Basic Access Mechanism) [From: Kurose01]

Even if it were Hidden Node and Fading Effects (discussed later) could still cause a collision. So a scheme known as collision avoidance was developed to overcome these limitations.

The primary medium access control (MAC) technique of 802.11 is called a distributed coordination function (DCF). DCF describes two techniques to employ for packet transmission to overcome the inherent Hidden Node and Fading limitations. The default DCF scheme is a two-way handshaking technique called basic access mechanism (Figure 7). This mechanism is characterized by the immediate transmission of a positive acknowledgement (ACK) by the destination station, upon successful reception of a packet transmitted by the sender station. Explicit transmission of an ACK is required since, in the wireless medium, a transmitter cannot determine if a packet is successfully

received by listening to its own transmission. In addition to the basic access, an optional four way hand-shaking technique, known as Request-To-Send/Clear-To-Send (RTS/CTS) mechanism (Figure 8).

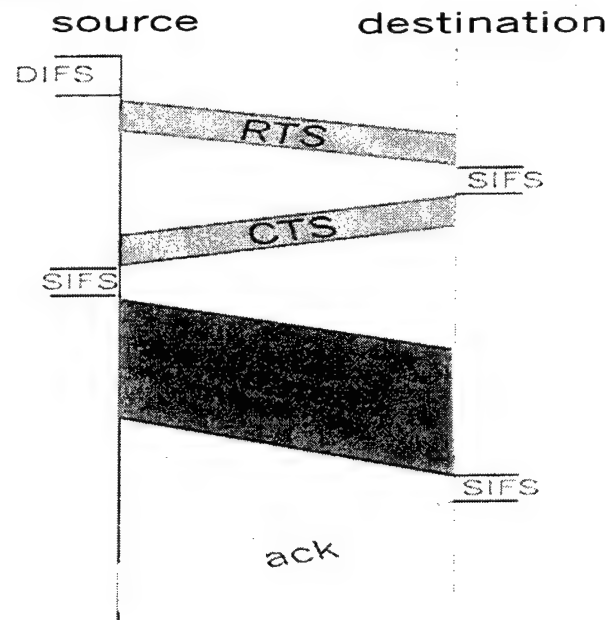


Figure 8 - Collision Avoidance (RTS/CTS) [From: Kurose01]

Before transmitting a packet, a station operating in RTS/CTS mode “reserves” the channel by sending a special Request-To-Send short frame. The destination station acknowledges the receipt of an RTS frame by sending back a Clear-To-Send frame, after which normal packet transmission and ACK response occurs. Since a collision may occur only on the RTS frame, and it is detected by the lack of CTS response, the RTS/CTS mechanism allows an increase in the system performance by reducing the duration of a collision when long messages are transmitted.

A station with a new packet to transmit monitors the channel activity. If the channel is idle for a period of time equal to a Distributed Interframe Space (DIFS), the station transmits. Otherwise, if the channel is sensed busy (either immediately or during the DIFS), the station persists to monitor the channel until it is measured idle for a DIFS. At this point, the station generates a random backoff interval before transmitting (this is the collision avoidance feature of the protocol), to minimize the probability of collision

with packets being transmitted by other stations. In addition, to avoid channel capture, a station must wait a random backoff time between two consecutive new packet transmissions, even if the medium is sensed idle in the DIFS time.

Since the CSMA/CA does not rely on the capability of the stations to detect a collision by hearing their own transmission, the destination station transmits an ACK to signal the successful packet reception. The ACK is immediately transmitted at the end of the packet, after a period of time called Short Interframe Space (SIFS). As the SIFS (plus the propagation delay) is shorter than a DIFS, no other station is able to detect the channel idle for a DIFS until the end of the ACK. If the transmitting station does not receive the ACK within a specified ACK Timeout, or it detects the transmission of a different packet on the channel, it reschedules the packet transmission according to the given backoff rules.

As mentioned before collision avoidance is implemented by the DCF in two ways the Basic Access Mechanism and optional RTS/CTS. These methods will help prevent the two problems that can occur in wireless systems.

a. Hidden Node

The hidden node case (Figure 9a) arises when station A transmits to B. C is also transmitting to B. A hidden node occurs when a physical obstruction prevents A and C from hearing each other's transmission, even though both are interfering at destination B. The hidden node effect causes an undetectable collision.

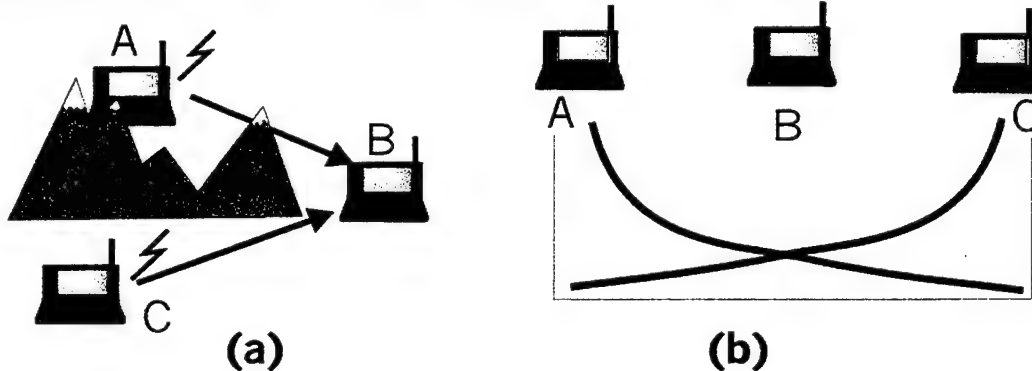


Figure 9 - Hidden Node (a) and Fading Effects (b) [From: Kurose01]

b. Fading Effects

Another undetectable collision occurs as signal strength fades as it propagates through a wireless medium, known as a fading effect (Figure 9b). Station A and C are placed such that their signal strengths are not strong enough for them to detect each other's transmission. However, they are strong enough to interfere with each other at station B.

4. Security ²

Effective security policies are developed on all levels of the OSI stack. Operating systems like Windows NT has an elaborate security model that centers around access tokens that are created when users perform valid logins. These access tokens provide the keys to resources within the network. Then there are other lower level security options that deal with the communication channels themselves. Communication channel protection will be the focus in the following discussion.

The Internet itself is an unprotected communications channel. It is vulnerable to hackers and eavesdroppers that can intercept and alter data. Packet sniffer programs can intercept packets and reassemble and analyze them quite easily. Several advances in digital cryptography have helped safeguard against these types of intrusions. These advances can be broken down into several categories [IDS00].

The first is Symmetric Cryptography. Here a secret key is used to encrypt and decrypt digital data. This encryption technique can very quickly encrypt and decrypt large amounts of data. Some of the more popular Symmetric Cryptography techniques are 56-bit Digital Encryption Standard (DES), 112-bit Triple DES, 40-bit Exportable DES and 128-bit Blowfish. A big disadvantage of this type of cryptography is that both the sender and receiver must possess the secret key to allow use of this secure channel. The management of this key then poses a security risk and administrative burden.

The second type is Public-Key Cryptography. Both parties have a paired private and public key. The public key is made public so that anyone can obtain it. The private

key is held and showed only be known by its user. This cryptography helps minimize the security burdens involved with Symmetric Cryptography and not only gives a means to encrypt data it also allows a sender to digitally sign a message. In effect it allows authentication and verification of sent messages. A short example illustrates this. If John wants to send an encrypted message to Nancy, he encrypts it with Nancy's Public key. Only Nancy has her private key pair and is the only one who could decrypt the sent message. If John wants to digitally sign it, then he would perform the above encryption and also perform a second encryption with his private key. This signs it. Since Nancy has access to John's private key, she can decrypt and she knows it is from John. Then she can decrypt with her private key to ensure the message is authentic (i.e. not tampered). RSA is one of the most popular Public Key techniques. A disadvantage to this type of cryptography is the "Man in the Middle" attack that could intercept and deceive a sender and receiver by replacing a public key during a public key swap between a sender and receiver. This type of attack is countered by having a trusted third party known as a Certification Authority to ensure safe transport of public keys. Another disadvantage of Public-Key Cryptography is the algorithm is a lot slower and more complex than Symmetric Cryptography and leads to what is called Hybrid Cryptography.

The third is Hybrid Cryptograph. This involves using both Symmetric and Public-Key Cryptography. Symmetric is used for high speed and large file encryptions and the Public-Key is used to securely transfer the Symmetric keys between parties. This is a more practical approach to cryptography and is implemented in what is called Secure Socket Layers (SSL). This client/server protocol is a widely accepted method of creating secure channels. It was developed by Netscape and is implemented automatically in most Web browsers and Servers.

D. **HARDWARE^{ALL}**

A lot of time and effort has been used to perform detailed and effective evaluations of wireless related COTS hardware components. An overview is given of

the criteria that were looked at when performing these evaluations. Also presented is a simple summary of the more effective devices that were encountered during past research.

1. Handheld Devices^{All}

When evaluating handheld devices for shipboard use a lot of factors need to be considered. Those include price, pen and input options, handwriting recognition, keyboard options, voice input, battery life and consumption control, expansion and card slots for PCMCIA cards, size and ruggedness, processor speed, RAM size and disk size/type, type operating system, Java JVM available, color and screen resolution, and ease of driver upgrade and Operating System (OS) reinstalls.

There are a variety of operating systems available into today's handhelds. They include PALM OS (C/C++ language), Windows 9X, Win CE 2.0, Linux, etc. Of course in our evaluation the Win 9X, was the easiest to work with because it is the defacto standard in OS. That is the availability of drivers, ease of installation, and applications available all favor this OS. It also has a full-featured browser with a built-in Java Virtual Machine (JVM) that allows the applications that have been written to run. Of course, the drawback to Windows 9X is the reboot process that is required each time you have to reconfigure the network drivers or TCP/IP settings. Win CE 2.0 was better in some respects because it is an ON/OFF device. It does not boot through MS-DOS. The PALM has the same type feature. The drawback to PALM and Win CE 2.0 is the limited applications that are available to interface in a network environment. Also the lack of Java JVM, made those platforms undesirable in development of Shipboard applications. However, technology is changing quickly and that hurdle is quickly becoming old news.

Mitsubishi Amity VP^{2,3,4,6} and Cassiopeia PA-2400⁵ were a couple of the many handhelds that were evaluated. The Amity VP is a Win 9x platform with a 133 MHz AMD, Am5x86 processor. It has 48 MB of memory and 810 MB hard disk. The screen is 7.5" color VGA. It also has two Type II PC card slots accepting many types of PCMCIA cards. The PA-2400 is a Win CE 2.0 device that runs an 80 MHz, Hitachi S-3

Super-H RISC microprocessor with 16 MB of ROM and 8 MB of RAM. It has one Type II PCMCIA slot, one Type I CompactFlash™ slot, and one Infrared port. Pictures are listed in Figure 10 with evaluation listed in Figure 11.

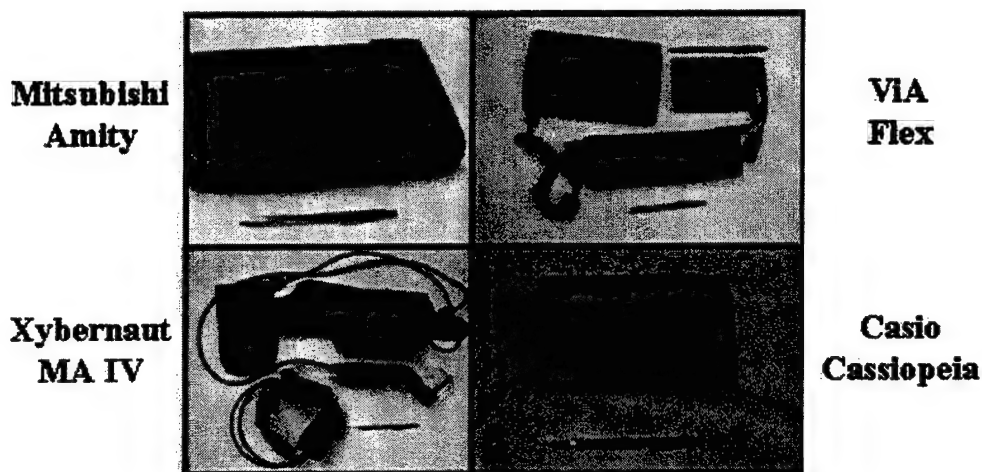


Figure 10 - Handheld and Wearable Devices That Were Evaluated

2. Wearable Devices^{2,3,4,5}

Wearable devices include the Via II Flex^{2,3,4,5} and Xybernaut MA IV^{2,3}. The Via II Flex has a flexible MediaGX processor from Cyrix that runs at 180, 200 and 233 MHz. It weighs only 22 oz; runs Win 9X and has a 3.2 GB hard drive with 64 MB of DRAM and two Type II PC card slots. It has Lithium hot-swappable batteries that can run for 6 hours.

The Xybernaut has a 200Mhz processor, a 2Gb hard drive and runs Window 9X. It has two types of displays: a monacle or a screen. The monacle is a computer image that allows hands free operation.

3. Performance of Above Tested COTS Equipment^{1,4,5}

Figure 11 shows the results of the evaluation. Amity performed as the best overall handheld and the Flex performed as the best wearable device.

Device	Expense	Durability/ Ergonomics	Functionality	Battery Endurance	Interface
Amity	Medium	Good/Average	Good	Average	Average
Flex	High	Average/Good	Good	Good	Average
Xybernaut	High	Average/Poor	Good	Good	Poor
Cassiopeia	Low	Good/Good	Poor	Not Measured	Average

Figure 11 - Evaluation of Selected COTS Equipment

4. Wireless Components^{All}

When evaluating wireless components the research was specifically targeted at ones that can create a wireless network. Those include PCMCIA devices, Access Points, and ISA cards. The criteria use to judge those components are: Speed, IEEE 802.11 compliance, output power, range, bandwidth, ease of installation and configuration, and available antenna attachments.

Commercial components are grouped into FH/SS or DS/SS devices that are non-802.11 and 802.11 compliant (Figure 12). Testing has been on components that operate in the 900MHZ¹ and 2.4 GHZ range. The general market shift has been towards the 2.4 GHZ standard and thus has been the focus of more recent research. Some of the 2.4Ghz basic service sets were Proxim RangeLAN802 (FH/SS, non-802.11)^{2,3,4,5}, Aironet 4500/4800 (DS/SS, 802.11)⁶, BreezeCOMA-10 (FH/SS, 802.11)^{2,4}, and Lucent WaveLAN (DS/SS, both non 802.11 and 802.11)^{2,3,4,6}. The figure below shows the basic service sets (AP's and PCMCIA cards) of the ones listed above. During most of the test the LUCENT WaveLAN (DS/SS, 802.11) was the winner based on upgrade capability, speed, and function ability.

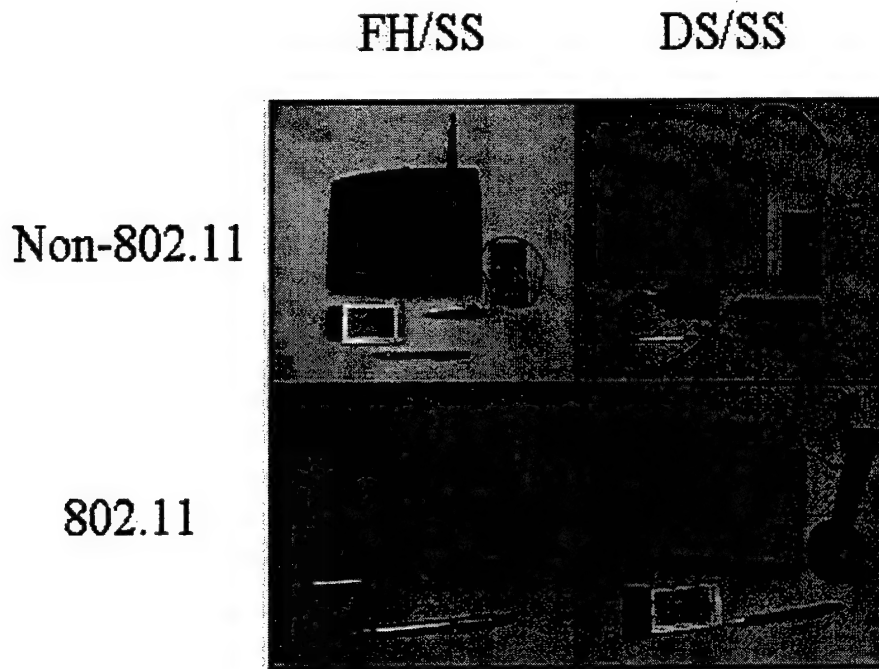


Figure 12 - Wireless Components That Were Evaluated

E. SOFTWARE DEVELOPMENT^{3,5}

1. Coding Languages^{3,5}

The programming language used for all prior developed shipboard applications was Java by Sun. It was used for a variety of reasons. It is a write once run anywhere language because it uses an intermediate structure called a byte code, see Figure 13. This byte code can run on any Operating System (e.g. Windows 9X, Windows NT, Linux, Unix, etc) as long as there is an interpreter for that type of platform. This interpreter is known as a Java Virtual Machine (JVM) and can run in a stand-alone mode or within a web browser. More specifically, Java source code (.java file) is intermediately compiled to byte code (.class file). This byte code is loaded and verified by the JVM. Then the bytecode is interpreted to run as an application or applet running over a network. Sun has recently introduced a Just in Time (JIT) compiler. A JIT

compiler translates and stores the entire class file during loading this provides faster overall execution of the Java bytecode because it avoids the program having to interpret each line of code, which leads to performance hits if a section of code is executed multiple times.

Sun advertises that Java is platform independent, allows for rapid development, has a large class library, object oriented, high performance, near real time, multithreaded, simpler and less bugs than C++, secure, and designed especially for networked environments. We have found this to be true in most cases. The performance of Java compared to C++ applications was as good as 100% using a JIT compiler [Mangione98]. Interpreted bytecode (older method) caused Java to run 4-5 times slower than C++ [Mangione98]. Also, Java has a built in garbage collector that dynamically scavenges memory references automatically, which makes programming easier but does create some real time dependency when the application executes. Besides these performance issues, Java has proved successful for the test of concept programs that have been developed so far.

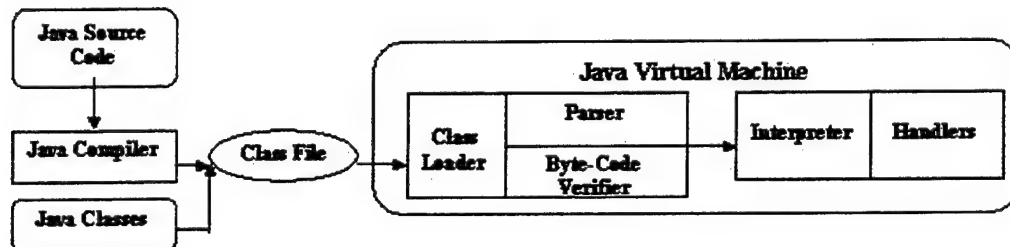


Figure 13 - Java Virtual Machine Flow Path

2. Feasibility Application^{3,5}

The applications developed were a Damage Control Client system for a surface ship and a submarine. Also log taking client applications were developed that connected

to a Database (MS Access) to prove that Java can connect and update a database in a wireless system. These applications still need a Damage Control Central console to make a working system that is implementable onboard a submarine. More detail in this area will be presented during the redesign described in Chapter III.

F. TESTING^{ALL}

1. Platform Environments^{All}

a. USS Ohio Test (August/1997)¹

This test gave data on how well wireless components work on a 726 class submarine. This test included a Digital Ocean Grouper wireless LAN (900MHZ, DS/SS), and a Newton MessagePad 2000. Some of the results worth noting were a 70 to 90 kbps throughput. It was also observed that transmission at 2050 mw caused some EMI spicks on meters in the Engine Room.

b. USS Harry S. Truman Hangerbay (March/1999)^{3,4,5}

This test gave some interesting insight into wireless testing in a hanger bay (Figure 14) of a good size ship. It was conducted at Newport News in a Hanger Bay that contained no Aircraft but did contain some availability support equipment. Equipment used was a Lucent WavePoint II AP and PCMCIA cards which were 2.4 Ghz, DS/SS, 802.11 compliant running at 2 Mbps. Also some testing of the Via II and Xybernaut wearable PC's were conducted. An Amity handheld and Dell

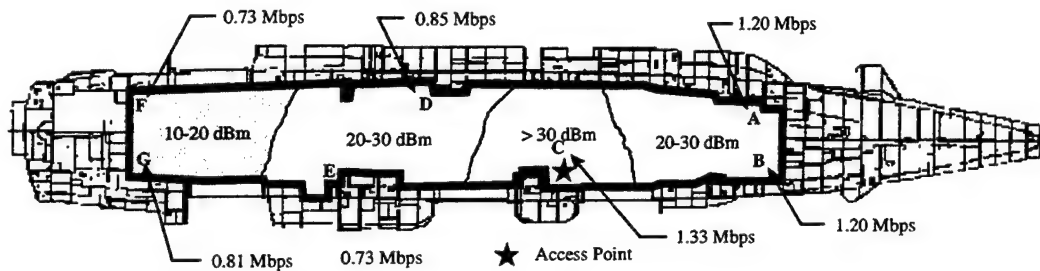


Figure 14 - USS Harry S. Truman Hangerbay

Laptop were also used. Below shows the area of coverage and the data rates seen as clients were roaming off of one access point throughout the hanger bay. Figure 15 shows the throughput as the clients were increased.

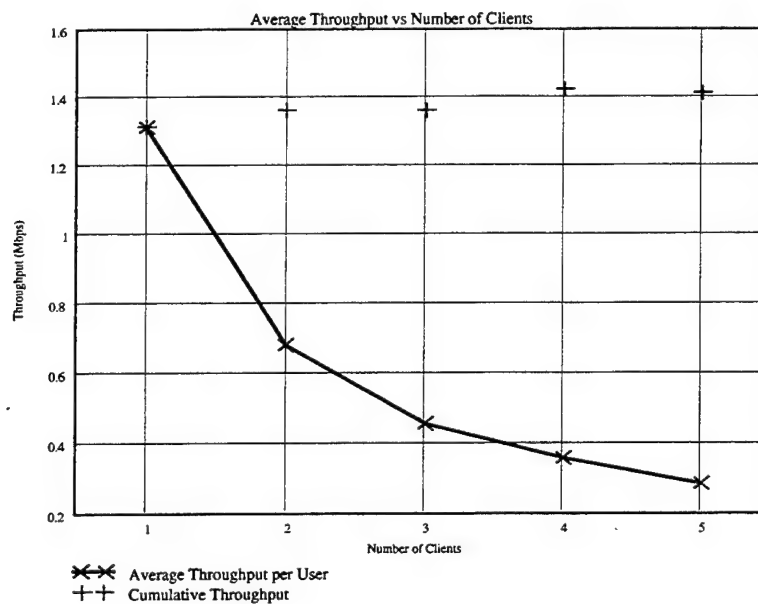


Figure 15 - Average Throughput vs. Number Clients (USS Truman)

c. USS Memphis (August/1999) ^{4,6}

This test was done on a platform that is designated for testing. Client/Server data rates were collected sending FTP files in the Forward compartment and Engine Room. For a 2MBps system the throughputs that were measured averaged around 1.3 Mbps for a single client. It was also shown that 10 access points provided full coverage throughout the submarine as shown in Figure 16. Also DS/SS performed well in a Multipath environment. The ship diagram was not included due classification requirements.

FORWARD COMPARTMENT		
Access Point	Space	Location
AP 1	FCML	Forward End of Crew's Mess
AP 2	FCML	Aft of Central Air Monitoring Station
AP 3	FCLL	Stbd Side of Auxiliary Machinery Room
AP 4	FCLL	Forward End of Torpedo Room
AP 5	FCUL	Stbd Side of Combat Systems Electronics Space
AP 6	FCUL	Aft End of Control on Port Side
Average Throughput		1.31 Mbps
ENGINE ROOM		
Access Point	Space	Location
AP 7	ERMLA	Forward End of Shaft Alley, Port of Centerline
AP 8	ERLL	Forward Part of PLO Sump, Stbd of Centerline
AP 9	ERUL	Forward, Stbd Corner of ME Bedplate
AP 10	ERMLF	Centerline, Above Vital AC Switchboards
Average Throughput		1.34 Mbps

Figure 16 - Access Point and Throughput Coverage (USS Memphis)

d. NPS Labs (3/2000) ^{2,4,6}

This testing was done to compare data rates between Lucent WaveLAN and Aironet 4800 series Turbo PCMCIA Cards. Both are 2.4Ghz and rated at 11Mbps.

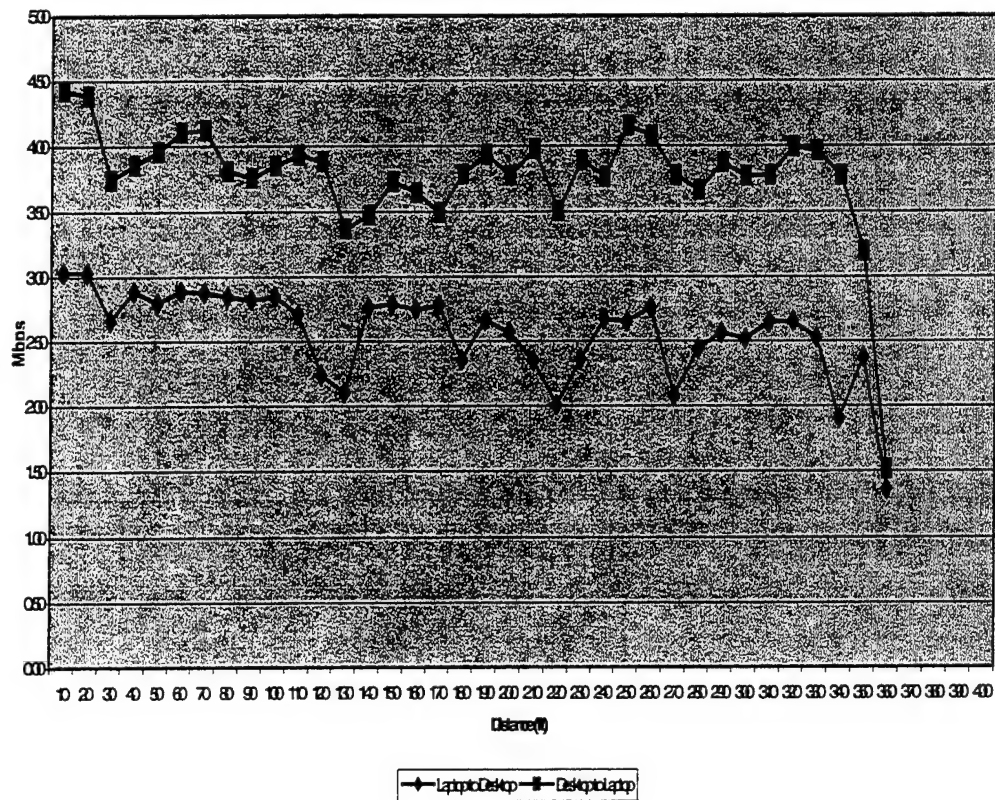


Figure 17 - Lucent WaveLAN (Turbo) Line Of Sight

Lucent's card is 802.11b compliant and Aironet is 802.11 compliant. The test was performed in a direct Line-Of-Sight (LOS) and non-LOS mode. The results from one of the test are shown. The data rates seen were 2.0 to 4.5 Mbps and represent 100% improvement over previous generation products. Below is a sample Line of Sight graph for the Lucent WaveLAN Turbo in both directions. It shows that max distance is about 330 feet before a significant drop in data rate is seen. Lucent performed slightly better in overall average data rates compared to Aironet.

G. SYSTEM REQUIREMENTS ¹

1. Requirements ¹

The actual requirements are subject to change as hardware gets better and prices drop on available COTS technologies. Figure 18 shows the typical cost required to outfit a wireless subsystem onboard a Virginia Class Submarine. These figures were looked at to further analyze the feasibility for deployment with regards to budget management.

Description	Number of Units	Cost per Unit	Total Cost
Handheld Computers (plus wireless pack)	14	\$200	\$28,000
Access Points	10	\$299	\$2,999
Server	1	\$2,500	\$2,500
Miscellaneous	N/A	N/A	\$10,000
SYSTEM TOTAL COST			\$33,499

Figure 18 - System Cost Wireless (Subsystem)

H. CHAPTER SUMMARY

During the review of the past research it has been determine that some areas are important enough to warrant future research. Those areas include the development of applications that take advantage of wireless technology. Those applications would include Damage Control interface and log taking software. A network analysis is needed on the proposed Virginia class wireless network to ensure that the applications created can be supported within this network. There are many tools such as OPNET Modeler that can do this. Also, other areas that need more research are concerns about

Security and Electromagnetic Interference (EMI) affects. Also, a look at the new Bluetooth technology that allows small Personal Area Networks (PANs) to connect components wirelessly when less than 10 feet shows promise when used with the current sensor technology. The HiperLAN standard should also be looked at because it promises higher wireless bandwidths to be delivered from server to client. Based upon this review of past research, current members of SWLAN project have set out to further research Security, EMI, Bluetooth, HiperLAN, DC Applications, and Network analysis to provide more assistance to NAVSEA PMS450. Chapters III, IV, and V of this thesis concentrate on designing a new type of DC application. Chapter VI performs a network analysis of the Virginia class Non Tactical Data Processing System (NTDPS) to ensure it can handle a load presented by this new application.

III. APPLICATION AND NETWORK REQUIREMENTS

Networks provide for multiple and redundant ways to communicate. This chapter explores some key network issues and concepts that relate to the remainder of this thesis. It is designed to lay the groundwork for building a vital submarine application known as Damage Control (DC). First, key concepts such as the TCP/IP model are described to give the reader the basic framework for how networks really work. Then a description of socket-based communication is described to introduce the communication tools that can be used transport communications streams within a network. Next, this chapter describes the system that is trying to be modeled by showing a Damage Control status board currently used on USS Batfish. The chapter concludes by describing the necessary requirements a Virtual DC application needs in order to be implemented.

A. TCP/IP MODEL, THE WORKING MODEL, [HUGHES97]

The TCP/IP model, see Figure 19, is similar to the OSI model described in Chapter II, except it leaves out the session and presentation layers. These layers are not widely used in today's networks. In practice, application layer protocols generate streams of data that are sent to the transport layer. This layer encapsulates this data into segments with header information and is important because it provides some important transport layer functions, such as error control, flow control etc. that help make a network work properly.

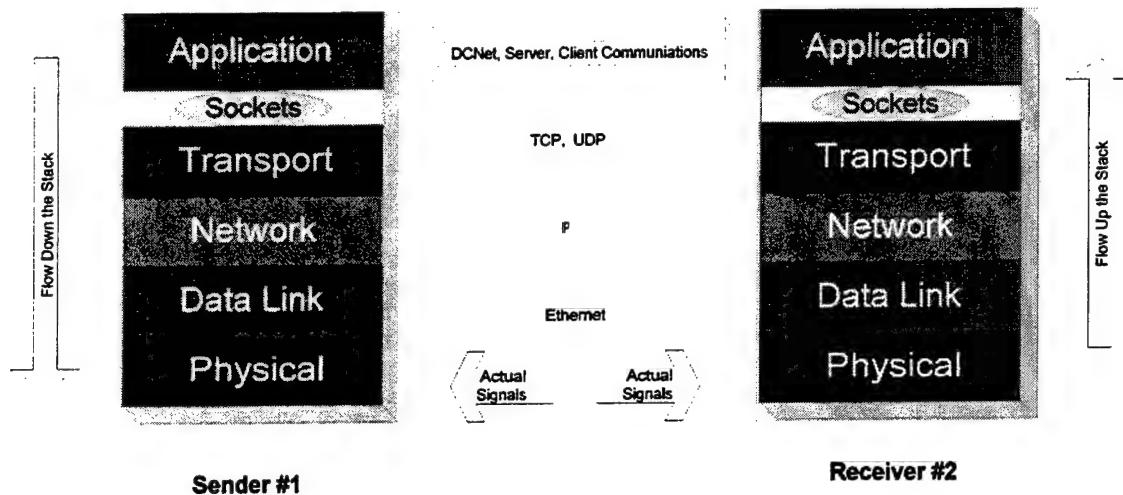


Figure 19 - TCP/IP Model

Two options are available in this layer, UDP (User Datagram Protocol) and TCP (Transmission Control Protocol). They both have advantages and disadvantages. Their use depends on the type of data that is sent and received. Both UDP and TCP send the segments they generate from the application layer to the transport layer. This layer puts these segments into IP packets and adds addressing information used to route these packets to a destination. The IP layer sends its packets to the data link layer, where the IP packets are encapsulated into frames, Ethernet Frames for this case. The Data Link layer also prepares the data to be put onto the physical cable, where it is transmitted as voltages.

B. INTERNET PROTOCOL (IP) ADDRESSING

IP addresses take the form of x.x.x.x, where x is one byte, represented in decimal format (see Table 1) byte has a range of 0000 0000 to 1111 1111 and thus has a range of 0-255. The maximum IP address is therefore 255.255.255.255. These addresses are uniquely assigned to each networked computer in the world and functions similar to a home address for mail delivery. The range of IP addresses are grouped into network classes based on the first byte:

Category	Address Range
CLASS A	1-127.x.x.x
CLASS B	128-191.x.x.x
CLASS C	192-223.x.x.x
CLASS D	224-239.x.x.x (Multicast)
CLASS E	240-255.x.x.x (Reserved)

Table 1 - IP Classes and Address Range

Classes A, B, and C are assigned to an organization based on balancing the subnet to hosts ratio. They provide a way to create a few numbers of subnets with many hosts, or vice versa. Class D networks operate in multicast mode and are the main transport address range that will be used to design a virtual DC application. Class E addresses are reserved for future use.

C. THE JAVA LANGUAGE

1. Why Java?

When designing submarine applications a look at the available languages is a necessity. C++, Java, Ada, and VB all perform some degree of network function ability. Based on ease of use, available libraries and power, Java has proven to be the designer's choice and used as a basis for most discussions and design during the rest of this thesis.

2. Security Issues

Chapter II addressed many of the encryption and network security issues. This section is to address some of the higher-level Java security issues. Java Application has full access to system resources, whereas programs developed as applets (downloaded from a server) work within a security Sandbox. This sandbox analyzes downloaded applets and determines if they are trusted or not. Trusted applets have digital signatures that may operate outside the sandbox. Outside the sandbox means you could download

an applet from a server and that applet could then have access to your local resources (i.e. file access, opening Sockets, creating threads, etc. The Sandbox rules are enforced by a Security Manager file that comes with all major java enabled browsers [Hughes97]. There are ways to override this security manager, but not without the users knowledge.

D. SOCKET BASED COMMUNICATION

1. What is a Socket?

Sockets are a basic component in networking software. They are a mechanism that operate at the transport layers and provide a mailbox for applications to send or receive data streams. These data streams then travel on the network and can hold virtually any type of data used in today's communications.

2. How Sockets Can Be Used for Communication

To operate a Socket in Java establishes a port number on the computer. This port number and the machine's pre-configured IP addresses are all that's needed to provide communications such as voice, video and data file transfer.

3. Submarine Applications and Socket Based Communication

There are many submarine applications that could benefit from networked applications, such as:

a. Drill Control

Using a drill team of shipboard personnel provides crew training during drill sets. This team initiates and controls each drill set applications that could be used to allow the drill team to effectively log the time performance of each drill set and provide instant feedback to the ship's Commanding Officer.

b. Supply System

This feature would allow petty officer to check for parts and order parts without leaving his workspace unattended.

c. Log Taking

Currently logs on machinery are written on paper. Trends of this data are usually just a simple review of the data by a supervisor. Integrating applications would allow the operator to log data to a database. This database could then be analyzed with more sophisticated tools to draw more concise conclusions from the data. The wireless feature of applications also gives the necessary freedom to the watchstander to take these logs.

d. Preventive Maintenance (PM)

All divisions on a Submarine perform weekly, monthly, quarterly, etc. maintenance on their gear. Applications would provide the technician with online maintenance procedures and provide an electronic log of successful/failed maintenance items. This log could then be used to track the corrections of the failed PM's.

e. Sub to Sub Data Transfers

Times could arise where submarines need to share data in a real-time, local environment. Applications could be used to link two-surfaced submarines (or two at periscope depth with a scope to scope link). They could then pass pertinent documents or lengthy conversations.

f. Casualty/Damage Control

This is where applications would truly shine. Communication is the hardest yet most important aspect that ensures effective casualty control. Applications would provide a central summary console to be located in control where the Commanding Officer would have access to real-time data. Important items like compartment rigs could be updated without clogging up phone circuits. Wireless transmissions and socket communication can travel through smoke and if access points were damaged by certain local hot spots, others would reliably transfer the important data to control.

g. Standard Operations

There are many day to day operations such as Ventilating the Ship or Shooting a Sonar Buoy that require multiple stations to interact. Applications could provide a socket based communication tool to allow more effective communication between operators and also allow access to the written procedures used in that operation.

h. Troubleshooting

When equipment breaks down, applications could be designed to include a powerful search engine to gather all pertinent documents at a touch of a button. This would minimize the hours it takes for a technician to remember what and where the needed documents are to fix this problem. This allows the technician to start troubleshooting sooner. The equipment could then be returned to a full ready status quicker.

This thesis selected Casualty/Damage Control concept for implementation.

4. Specific Socket Communications [Mahmoud00]

a. Transmission Control Protocol (TCP) Sockets

In Java, a TCP socket comes in two forms: A `serverSocket` and a `Socket`. A server socket waits for a client socket to request a connection. Once a connection is made, then a unique pair is created for data flow. TCP advantages include: guaranteed delivery, in order delivery, flow control, error control and congestion control. Its weakness is that it does not scale well. For each connection, N-1 open sockets are required (Connection-Oriented). It is similar to a telephone. Each friend you call requires a new connection. TCP also has a lot of overhead to provide its reliable servicing. This tends to make it slower than UDP. It also doesn't work well in a real-time or streaming environment. It does guarantee delivery, but there is no guarantee on when it will arrive. This best effort service is the way today's Internet is designed, though there is extensive research in areas to provide better service, including timing.

b. *User Datagram Protocol (UDP) Sockets*

UDP sockets do not form pairs like TCP. UDP just send data to an address on the network and hope it arrives. There is no guarantee for data delivery. That is its disadvantage. Its advantage is there is almost no overhead and only one open socket is necessary to send to multiple addresses (hosts). Therefore it scales better and is usually faster and better suited for real time and streaming applications.

c. *Broadcast Sockets*

Broadcast uses UDP sockets and sends messages on a special address. The special address allows the sender to send data once and all hosts on that network will receive it. This special address is the last number (255) within that network. (E.g. Class C net: 192.1.2.0 broadcast address is 192.1.2.255) It is better than UDP, because it doesn't require multiple sends for multiple hosts, only one send. Also, since it uses UDP sockets underneath, it only requires one open socket, whereas TCP requires a socket open for each host. The disadvantages are that all host receive the packets, which may not be desired.

d. *Multicast Sockets*

Multicast operates similar to broadcast, but it allows for some discrimination in who receives the packets. Also, whereas Broadcast has one address, multicast covers a range of addresses to choose from (224-239.x.x.x). Only host that have "subscribed" to an address will receive that data. Multicast works similar to 2JV phones on a submarine. One person talks, and if others are "dialed in" they can listen to the voice traffic.

e. *Remote Method Invocation (RMI)*

RMI is a Java based TCP implementation that provides a "remote calling" mechanism, that allows a host to make function or method calls on a server. This mechanism provides a way to make a system consisting of computers on a network seem like just one computer. In other words, it can hide the network from the application

developer. It can allow the developer to take advantage of a powerful server to perform task intensive code and provide output to the host, which may be slower.

f. Common Object Request Broker (CORBA)

CORBA is similar to Java's RMI, but it allows not only cross platform development, but also cross language development. RMI and CORBA are described here but not used in the design developed latter in this thesis.

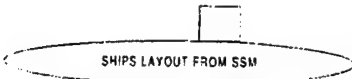
The Multicast method that was chosen to implement the design described in Chapter IV.

E. THE DC COMMUNICATION MODEL

Damage Control is the way casualties are fought onboard submarines. During damage control drills and actual casualties, several stations are manned. They include Damage Control Central (DCC), DC Fwd, DC Aft, and the Scene of the casualty. DCC is the main controlling station for the casualty with the DCC coordinator and assistant manning it. This station maintains the status via a grease pencil and status board similar to the USS Batfish status board depicted in Figure 20 [DCCConcepts96]. This status board is used to sort out fast moving details of the casualty. The DCC coordinator then makes reports and recommendations to the Officer of the Deck and the Commanding Officer.

The Scene is a party with a designated Man in Charge that is actively combating the Fire, Flooding, and other casualties. They make certain standard reports to DCC to make them and other supplemental watch stations aware of the current status.

BOAT: U.S.S. BATFISH
BOARD SIZE: 33" x 15"



CASUALTY 1		CASUALTY 2	
MAN IN CHARGE	MAN IN CHARGE		
ADDITIONAL PERSONNEL REQ	ADDITIONAL PERSONNEL REQ		
INJURED PERSONNEL	INJURED PERSONNEL		
DAMAGED EQUIPMENT	DAMAGED EQUIPMENT		
CASUALTY STATUS	CASUALTY STATUS		
REFLASH WATCH	REFLASH WATCH		
INITIAL SITUATION REPORT	INITIAL SITUATION REPORT		

NFTT		NAME		TIME	
NAME	TIME				
PDR ATM					
NAME	TIME				

CAMS ATM LIMITS		
1HR	24HRS	90DAY
O2 130-220		
H2 7.6		
CO 152.6		
CO2 30T		
R-12 1520MT		
R-114 1520MT		

RIG

SPACE	
CONTROL	
BOW	
DIESEL	
OPSUL	
OPSM	
TORP RM	
AMR 1	
AMR2UL	
AMR2LL	
ERUL	
ERLL	

POST FIRE ATMOSPHERE

Figure 6: USS BATFISH (SSN 681) DC Status Board

Figure 20 - Damage Control Status Board - USS Batfish

Other reports such as compartment rigs, atmospheres, repair and assistance team status, immediate and supplemental action status, engineering report are also sent. These reports are made on Sound powered phones throughout the ship and DCC writes their status on the grease status board. As you can see this casualty board tracks two casualties and maintains a ship rig status and atmosphere limits. Other such boards on other naval submarines function similarly. The boards are simple but nothing more than a notepad to organize information. A more helpful and dynamic solution is needed.

A DC software application can be used to emulate this process in a more efficient manner. First, the grease status board would be replaced with a 20" touch screen monitor designed to provide a graphical user interface that is laid out to give a DCC the maximum information. A GUI can layer information on top of each other, unlike a grease board. It also can update automatically or make decisions or recommendations

that based on the information received. This would serve to make the job of tracking information a lot easier for DCC.

Another advantage of this type application is the reduction of communications on the Sound Powered phones. Currently only one person can talk at a time, and sometimes a lot of reports are missed because DCC is reporting to the OOD or CO or updating the status board. A network can deliver multiple communications in both directions because of its duplex nature. Thus allowing rapid and multiple communications all at once. Currently failed communications are the top reasons why drills perform poorly during exercises done on submarines. Real casualties suffer from the same problems.

Another possible disadvantage is the use of a handheld computer during the heat of a casualty. Can we expect a person to be able to use a pointing stylus properly when there is smoke in his face or he is knee deep in water? The answer is probably not. Some type of video and voice system needs to be integrated within this system to help overcome this. These are issues that will be looked at later.

Another disadvantage this system would have is the susceptibility to a power outage that currently does not plague Sound Powered phones. A battery backup system would be needed to keep the network up during a loss of power. Also, a persistent storage model such as a backend database is needed to hold the state of the damage control status throughout the boat. This state would include the speed and current depth sensor, atmospheres and compartment rigs throughout the boat, all reports from casualty members and all supporting casualty reports. It should also track in what compartments the injured personnel are, the people wearing what Breathing Devices and time remaining. Hose teams are also important to track in a casualty. This information is some of the state information that should be kept and this concept should be integrated into the virtual grease board concept described earlier.

F. BUILDING A DAMAGE CONTROL APPLICATION

1. General Software Constraints

a. Scalability

There should be no imposed limits on the number of computers that can send reports. The application must scale easily to all members of a crew, which is typically up around 150 personnel.

b. Mobility

The user should be able to move around as the casualty is being fought. Wireless networks have this unique advantage because no wires are attached to the user.

c. Multithreaded

An application should multithreaded for a couple of reasons. First it allows a listening station to monitor more than one casualty and more than one user at the same time. Also, multithreaded applications tend to be faster since they allow parallel operations. Finally, listening sockets will block (halt that thread of execution) until a packet is received. If the listening socket is in the same thread as the GUI interface then that will lock the user interface until packets are received an undesirable effect.

d. Operate Similar to Current DC Communications

Multicast communications are similar to sound powered phone communications. Multicast works like a conference call. One or more people speak and others can listen.

e. Reliability Higher Than Current DC Communications

Multicast as discussed earlier works by sending UDP packets, which have no guarantees to arrive at their destination. However, dropping these packets at different routers onboard the submarine is highly unlikely because the congestion traffic would be low on the network. Several lab test conducted on reliable packet delivery have shown no lost packets conducted over several days of continuous transfer. In any case, the

reliability would be better than today's sound powered phone communications, where reports have been shown to be lost in the heat of a casualty.

f. Easy to Setup and Use

An application should be easy to use and setup once shipped for use to the submarine. There should also be adequate documentation to show an average sailor how to use and install it properly.

g. Network Capable

This application should be able to send and receive packet communication at any time and on any computer. This would allow anyone at anyplace to monitor the casualty, like maybe the CO in his stateroom. This portability is key to successful use of a DC system during a ship casualty.

h. Fast Data Transfer

Although not normally issue, if say video cameras are attached to NFTI or all users are voice capable then the network must have bandwidth to ensure the packets arrive at there destination in a timely manner. If they arrive late then it does someone no good when trying to fight a casualty in real time.

i. Customizable Configuration to the Ships Needs

Each ship will have different crew names. Standard messages such as Man in Charge or Injured personnel will be different for different ships. Also special equipment or slightly different approaches to fighting casualties are allowed to some degree onboard different submarines and the application would have to be customizable to reflect this.

j. Applet and/or Application Capable

These requirements would allow the software to be run within a browser served from Server on the network or ran as an installed application on that computer. Both have their advantages and disadvantages, but generally an application is less restrictive.

k. Persistence Protection

Recovery of the system to its current state during a power outage requires some type of persistent storage. Whether the state of the casualty is maintained on a DB or a text file system, this system needs to provide persistence protection if the networks and computers go down.

2. General Hardware Constraints

a. Open Operating System

System must support a common operating system (e.g. Windows 95/NT/CE, or Linux), however if written in Java this requirement can easily be relaxed because of multi-platform capability.

b. Network Connectivity

The virtual DC system must support constant and reliable network connectivity for communications.

c. Rugged

The system must be rugged enough for the afloat environment. However, it is important not to stress this too much, for it raises cost. If it's inexpensive enough and it breaks, you can just replace it.

d. Long Batter Life

System must have at least a few hours battery life, with the radio card active. System must also allow for a hot-swap of batteries without re-boot (Especially in Windows 95/NT models).

e. Mobile Input Method

System must support pen-based input, and either handwriting recognition or voice recognition to allow more flexibility for casualties that are restrictive to steam suits or fire fighting ensembles that can hamper pen-based inputs.

f. Mobile View Method

System must support a portable viewing method of either a head mounted monocular or sunlight viewable screen.

g. Comfort

The system must be comfortable in both form and function because of the long hours of use by crewmembers.

h. Storage

System must have suitable Read Access Memory (RAM) and hard drive capabilities to support full function applications.

G. CHAPTER SUMMARY

This chapter explored some of the specific network issues related to building submarine applications that can be used to demonstrate the usefulness of an onboard submarine wireless LAN. It explored some of the initial research for building a specific application. In the next chapter research is dedicated to building a Damage Control (DC) application and explaining how it works and its usefulness.

IV. SUBMARINE DC IMPLEMENTATION IN JAVA

This chapter sets out to describe a Java based application that can be used as a prototype to demonstrate the capability of a wireless US Navy submarine network. This description is intended to map the design process of this application. It also provides a comparison with the DC application designed in [Sayat99] to show why multicast communication is more reliable than the database centric approach taken in Sayat's thesis.

A. GOALS

SWIPNet stands for Submarine Wireless Prototyped Network. It's goals as a research topic are to investigate the feasibility of writing such an application in Java for a wireless network. As a deployed application it is designed to provide timely DC information to Damage Control Central (DCC) to combat casualties and to provide reports to the Officer of the Deck (OOD) used to maintain the Safety of the ship. It is intended to streamline communications of current DC practices by allowing messages to be sent and retrieved via wireless devices throughout the ship.

B. SCOPE OF THE PRODUCT

To provide a working application that can be easily installed on any type of platform, SWIPNet would consist of pen-based consoles that run Windows CE/95/98/NT, Unix, Linux, Palm or Macintosh to allow user input. These consoles would be bridged wirelessly to the ship's existing LAN via access points that are strategically placed throughout the ship for 100% connectivity. This wireless network would be used to take used in a variety of applications, namely damage control (DC). SWIPNet's distributed software would be written in Java to take advantage of its robust network class library.

C. FEATURES OF SWIPNET

1. Desired Features

Although all these features are not fully incorporated in the current design, it is setup so they can be easily added later in future iterations.

a. *Standard DC Reports*

To provide Standard Damage Control Reports for all major casualties onboard Submarines (Implemented).

b. *Voice and Video*

To provide voice and video communications to allow ship consoles throughout the ship to see the status at the scene without being there.

c. *Persistent Storage*

To provide persistent storage, ability to recover if power is lost (Partially Implemented).

d. *User Interface*

A user interface or console to send and retrieve reports that is familiar to current DC practices to minimize the learning curve involved (Implemented).

e. *Availability*

SWIPNet will be available 24 hrs a day connected to an AC adapter. It will also have a battery backup (up to 4hrs) with hot swap capabilities.

f. *Security*

Database and server will be locked and controlled by the SWIPNet Administrator.

g. *Encryption*

Applets version will run in Microsoft Internet Explorer with 128-bit encryption.

h. Maintainability

SWIPNet designed to function with minimal maintenance support. Using commercial of the shelf (COTS) products will allow for backup components (PDA's, server and database equipment at a low cost).

i. Customization

Available for different ship designs, e.g. Virginia, Los Angeles or Ohio classes. Different ship designs have different compartments and different names and types of equipment. This affects the standard reports that would be built into a virtual DC application.

j. Post Drill Feedback

Allow the system to email a report (or Database report) of times, actions and reports during a drill or actual casualty.

2. Key Design Features

a. Sender/Listener Approach

SWIPNet is based on a Sender/Listener approach. The Sender (Client object) in this case is the handheld computer that is deployed throughout the ship. The Listener (Server object) is designed as a bigger display that should be used by Damage Control Central. All roaming clients can send Damage Control reports to the Server, which functions as a listening station to receive and direct these reports to its appropriate GUI display. Future designs will incorporate sending features within the Server and listening features within the Client.

b. Multicast

Multicast was chosen as the primary communication method. Multicast sockets function similar to sound powered phone communications on naval submarines and ships. This communication allows some one to speak and if the station dial is tuned or dialed in to that station then they can hear the current communications. Any number

of people can hear these communications. The multicast socket is thus analogous to the sound powered phone station onboard the naval platform.

c. Applet/Application

Currently the application can run as an applet within a web browser. Some issues do arise. They concern security and Java's Sandbox protection scheme. Currently, if run as an applet the Sandbox will not allow the Client to start threads or open a multicast socket. These securities can be turned with Internet Explorer 5.0 or higher allowing the applet to run on the client like a preinstalled application.

D. OLD VS NEW COMPARISION

In [Sayat99] a DC network client was developed that used a database centric approach. This older version had a client applet (or sender) push the information to a database when an update was issued. Then the server applet (or listener) had to pull the information periodically to see the current state of the casualty. This thesis design took a different approach to the design by using multicast sockets wrapped within by a Java application to provide the communication channel. This effectively cut out the middleman, i.e. the database and provided a more reliable system. This new design is better for the following reasons:

1. Application is Better Than Applet

During a casualty it makes more sense to have a SWIPNet application already installed on a DC handheld. SWIPNet is designed to run as either an applet or an application. However, comparing the approaches in this research has shown the application approach to be more reliable and faster. Applets have sandbox restriction (discussed in Chapter III) that can be overridden, but this override tends to be browser dependent making them somewhat unreliable. Also, applets have to be served from a database server, which tends to slow the response during a casualty. The SWIPNet

application could be quickly downloaded and used, but DC equipment onboard submarines are always pre-staged making the preloaded application approach better.

2. No Bottlenecks

Having a database to hold the state is important but if the data must always pass through the database, then it tends to slow down the communication. Multicast sockets send directly to the listener and eliminate this bottleneck. This speed is very important in a casualty because reports must be generated in real time to allow the necessary actions to occur to control and stop the casualty.

3. Less Error Prone

The database centric approach is dependent on a JDBC-ODBC driver to allow the client and server to communicate with the database. This approach has a tendency to lock up and fail to update or retrieve from the database. This occurred with no error message making troubleshooting difficult. Also the clients and server consoles were designed as applets. Thus if the database server was down the applet would not be served.

4. No Single Point of Failure

The database centric approach is dependent on the machine the holds the database server or links to it to be fully operational for the whole system to even function. If the database server went down then the applet consoles and the state data could not be served. The approach taken in this thesis allows multiple machines to send reports via sockets and the information can travel along any routed path to listeners who have joined the group or channel making this system very fault tolerant.

5. Easier to Deploy to a Ship

The system built in this thesis is easily deployable to ships because it only requires install of a Java Runtime Engine, which makes the SWIPNet Jar file executable. The SWIPNet system will immediately start working over the network. The database centric approach taken in [Sayat99] requires a detailed setup of a database, a database

server (like IDS Server) and the ODBC driver, which could prove to complex for an average sailor.

6. More Scalable

The SWIPNet application designed in this thesis can scale to as many senders and listeners that are needed to combat the casualty. The database centric approach will only allow one client or server access to the database at one time to ensure integrity of the data. Thus more clients can communicate with the SWIPNet application and the setup process of the whole system would be easier.

There is one advantage that the database centric approach has over the SWIPNet design and that is the concept of persistent storage. The database holds the state in case of a power outage onboard a ship. This makes integration of a database as a state holder and system initializer important. Based on this the concept will be integrated into the SWIPNet design, but not at the expense of speed.

E. SYSTEM SPECIFICATIONS

1. Package Layout

The SWIPNet design is laid out into five different packages (control, dcObjects, gui, shipObjects, and utility). These packages group the similar class files used in this design. Table 2 describes these packages. The control package holds the main interfaces that run this class. Each dcObject and shipObjects contains a gui object from the gui package. Gui classes were separated into their own package to allow easier modification to the interface.

Packages	
swipNet.control	This package contains DCnet, Server, and Client, which are used as the main interface and controls for the user.
swipNet.dcObjects	This package contains the damage control Objects used by the control package to hold dc reporting information.
swipNet.gui	This package contains all GUI parts implemented by swipNet.dcObjects and swipNet.shipObjects and package. They are separated from the shipObjects and dcObjects, to allow editing of the user interface independently of lower level code changes.
swipNet.shipObjects	This package contains the ship Objects used by the swipNet.control package to hold ship-reporting information.
swipNet.utility	This package contains utility functions and classes used throughout the project, like PostOffice, Initialize and JDBCBridge (not fully implemented).

Table 2 - SWIPNet Package Summary

Figure 21 shows the relationships between each of the packages and the classes they contain. DCNet is the main interface. It configures and launches a client and/or server. The Client object sends dcObjects and shipObjects via multicast packets generated in the PostOffice class. The Server then receives these packets using its Post Office from the network. These packets contain the pieces of the dcObject and shipObjects broken up into network bytes. The Initialize class provides all standard DC reports needed to initialize the system. Also, from this figure it shows that each gui classes is part of its respective dcObject or shipObjects. The JDBCBridge class is not shown because it was not fully implemented. However its intention is to allow a persistent database to provide standard DC inputs to the Initialize object and have this object to travel the network and initialize each client and server (see Recommendations for Future Work in Chapter VII).

Fully Implemented Class Summary

Client	Client subscribes to two multicast address and sends Casualty and Ship reports within one Communicate thread; It creates a client GUI by adding GUI objects such as FireGui, etc and uses a PostOffice object to send, Fire, Flooding objects, etc.
DCNet	DCNet class provides and configuration and launching display, it has the ability to configure the station location, multicast address and launch a Client (sender) and/or Server (listener)
Server	Server is a listener object, it subscribes to three multicast address and receives two casualty objects, and ship objects; All three run within 3 communicate threads; Server creates its on GUI objects and uses PostOffice to receive sent packets
Fire	A Damage Control casualty object that holds information within strings that can be directly sent over the network for Fire casualties; The fire object is fully implemented and fully documented.
FireGui	Graphical user interface (GUI) that shows either Server or Client components and represents the data contained within it matching dc or ship object; GUI isolated by itself to allow easier editing
ShipStatus	A ship object that holds information within strings that can be directly sent over the network for ShipAtmospheres parameters
ShipStatusGui	Graphical user interface (GUI) that shows either Server or Client components and represents the data contained within it matching dc or ship object; GUI isolated by itself to allow easier editing; ShipStatus contains some, extra threads (w/ depreciated methods) to simulate depth and speed on a progress bar
Initialize	A class that holds information for Initializing all objects, isolated to its own class to allow for future development with a initializing database and allow reuse of its String array components
PostOffice	Responsible for sending and receiving of streams on designated network multicast addresses

Key SWIPNet Class Methods

DCNet Class	
<code>main (String[] args)</code>	Entry Point, used to initialize the applet as an application, allows program to function as an applet or an application
Server and Client Class	
<code>init()</code>	<code>init ()</code> - nest <code>jbinit ()</code> , to catch exception and use Jbuilders GUI builder interface
<code>start ()</code>	<code>start()</code> , starts the casualty and ship threads
Client Class Only	
<code>returnCurrentTime()</code>	Utility method to parses the time from an entire date object. This method is private to the Client class
Fire, Flooding, ShipStatus etc Classes	
<code>setOwner(String A, String B)</code>	Set the ownerName and multicast address A - ownerName, like DCC; B - multicast you are sending on
<code>setData(String A, String B, String C)</code>	A - IP address and domain name of the computer that's using this object; B - time Sent stamp; C - any message to add
<code>setStatusFromGui()</code>	When called takes the current Combo Box settings and copies it to its corresponding String
<code>copyStatus(Fire fire)</code>	Takes the Strings from the callers Fire object and copies to its matching text field
<code>Fire(byte[] aBuffer)</code>	This constructor allows casting to the appropriate object depending on the its type; The <code>DataInputStream</code> should be read in the SAME order that the <code>toBytes</code> places them onto the <code>ByteArrayOutputStream</code> , ORDER Counts

toBytes()	This method converts or writes all instance variables within the fire object to a stream to allow a byte buffer to be sent on the network, again ORDER Counts
FireGui, FloodingGui Classes, etc	
FireGui(boolean isServer)	Fire() Constructor, allows this class to be used for Client or Server; Their GUIs are different. isServer - if true, initialize as a Server, if not then display as a Client.
PostOffice Class	
sendMulticastPacket(Object typeObject, MulticastSocket socket, InetAddress address)	This method takes Objects like Fire, ShipStatus, etc, a Datagram (or multicast) Packet by using that object toBytes() method; It assumes that a the Socket has already joined a multicast Address.
receiveMulticastPacket (MulticastSocket socket,InetAddress address)	This method receives Objects like Fire, ShipStatus, etc and converts it to the appropriate object so the caller can use "instance of" to determine the correct object.
factory (byte[] ba)	This method accepts a byte[] array and looks at the first int value to determine what to cast the object to. Each casualty or ship object like Fire, ShipStatus etc. has an instance variable (typeOfDCObject), which is always ordered First. This Method returns an Object that holds the disguised DC casualty or ship object

Table 4 - Key SWIPNet Method Summary

F. GRAPHICAL USER INTERFACE (GUI) LAYOUT

1. In General

a. Color Use

Colors are used within each panel of the Launching console (DCNet object), Sender (Client object) and Listener (Server object). These colors are used to provide color recognition and separation within each panel.

b. Compactness

The Launching console (Figure 23) is designed to be small and unobtrusive. The Listening console is bigger to allow more information to be displayed at once. This allows for two casualties and ship status to be monitored at the same time in three separate panels. The Listening (Figure 25) console also has a Station Reports and Ship Control message area to determine what station has just reported. Also, the Listening console has a built in submarine diagram to help track casualty locations. Another feature the Listener has is the ability to slide unused panels out of the way. The Listening console is designed for a 18" touch screen. The Sending console (Figure 24) is designed for a small handheld wireless connected console. It lists tabs for all dcObject GUI's and shipObjects GUI's for compactness. These objects are not separated as in the Listening console.

c. GUI Components

The Java Swing class is used throughout this design. Specific components include, JComboBoxes (for listing items), JLabels, JProgressBars, JPanels, JTextfields, JTextAreas, JScrollablePanels, JRadioButtons, JButtons, JToggleButton, JFrames and JApplets. Future design will include more components to maximize the information displayed.

2. Launching (DCNet object) Console

The Launching Console (Figure 23) contains multiple labels and combo boxes designed to allow the user to select pertinent initialization data. The DCNet will configure and launch a Sending Console via the "Send DC Channel Reports". The Location and Multicast channels are needed before the Sending console is started. The "Set Location" is the station that the Sender is located at onboard the ship. The Set DC Channel and Ship Channel set the multicast address, which the different reports will be sent on, for that specific casualty and the ship status reports. The "Monitor DC Channels" launches a Listening Console and configuration functions similarly to Reporting console except it has an extra combo box to set a second DC multicast

address, for monitoring a second casualty. A hide feature is used to place the respective GUI in the background.

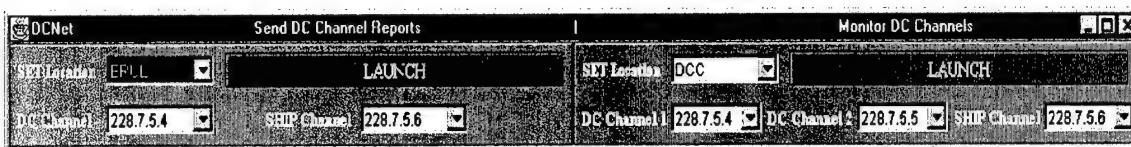


Figure 23 - Launching (DCNet object) Console

3. Sending (Client object) Console

The Sending GUI's (Figure 24) location in this case has been set to ERUL, and is sending the Fire reports on the 229.7.5.4 multicast address. If the Server is listening on this address then the reports will be heard and displayed. If the Ship tab is selected then the address shows 228.7.5.6, and is the channel those reports are sent on.

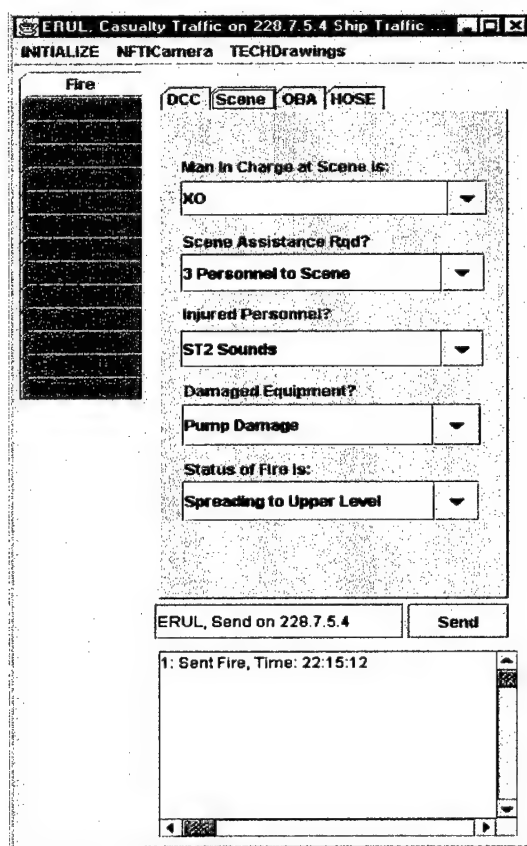


Figure 24 - Sending (Client object) Console

Future iteration will design the ability to receive into this console. Multiple Clients can run on the same machine, making it possible to send on more than one multicast channel, and thus each client could respond to more than one casualty. The console is laid out to maximize the amount of potential displayed information in the smallest space.

4. Listening GUI (Server object) Console

The Server objects at this stage in the design are just listening windows (Figure 25). Future development will design in the ability to send also. This example shows the location set to Damage Control Central (DCC).

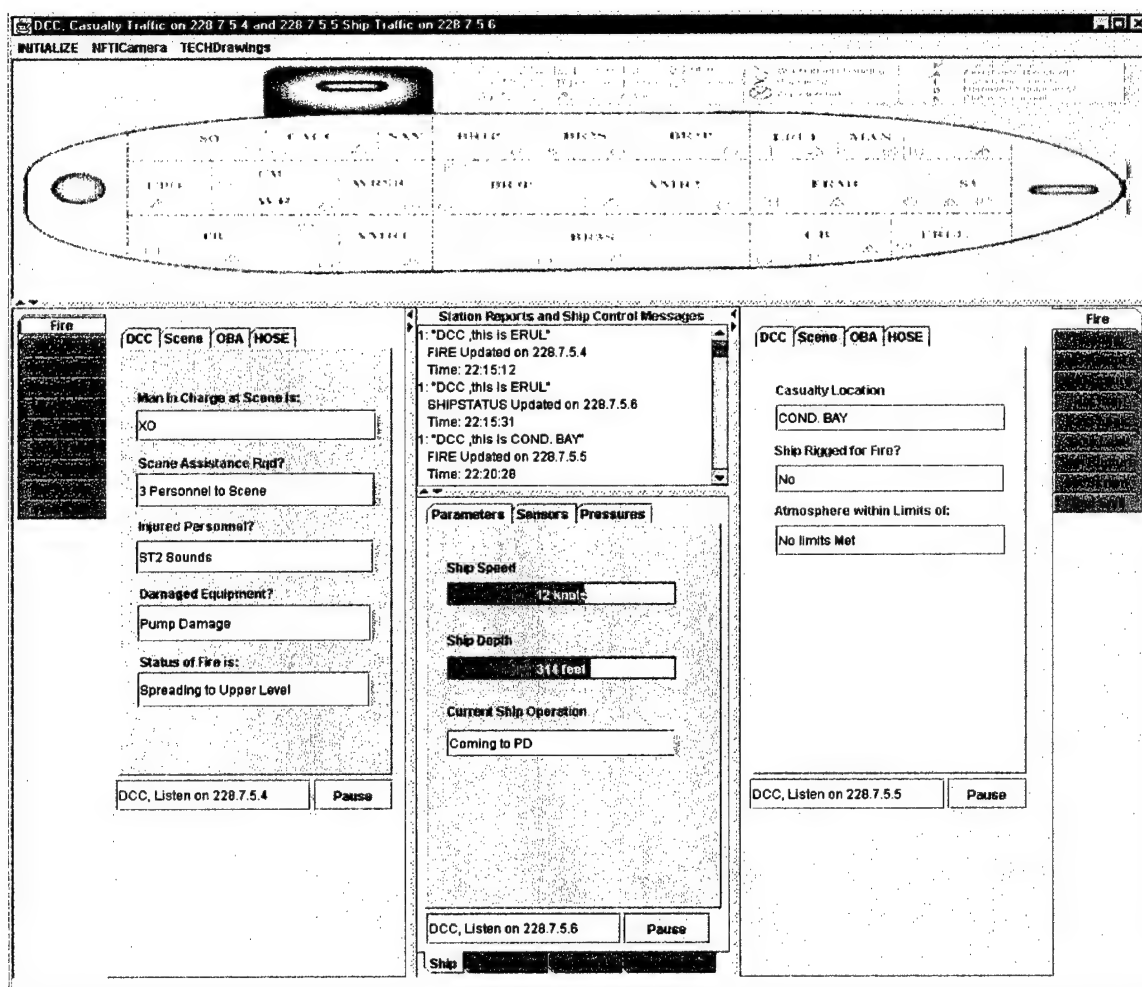


Figure 25 - Listening GUI (Server object) Console

The center panel displays reports for Ship, Engineering, Compartment Rigs, and Atmospheres. This information is central to both casualties and thus located in the center. The Message window receives and displays reports sent from different stations throughout the ship.

5. Deploying the Application for Use

Jar File Execution – A nice feature of Java is it allows a developer to compact all the source code in a deliverable package known as a jar file. This allows a user to have the ability to run the application on any machine because all files necessary to run them are available. The current jar file for the developed SWIPNet application is 161 KB. This size makes for a fast download to any networked computer in the network. To run the application all the user needs is a Java Virtual Machine which can be downloaded free from Sun Microsystems within the Java 2 Runtime Environment (JRE). To ensure a jar file can be executed when clicked by the user an entry into the jar's manifest file is needed to identify the main () class in your project. The entry would be similar to Manifest-Version 1.0, Main-Class: swipNet.control.DcNet. The JRE, when installed, creates an association with the .jar extension to allow the operating system to associate the JVM and your main class in your project

G. CHAPTER SUMMARY

This chapter set out to develop applications for a submarine wireless network. Although there are many possible applications that could be developed, Damage Control was chosen based on its complexity and need for improvement. With this application, it was shown that multicast communication is a powerful way of transmitting multiple traffic streams in a quick and efficient manner. It also mimics well they current way communications are down on a submarine. This chapter described the advantages the multicast application approach has over a database-centric, applet-type approach designed in [Sayat99]. The applications presented in this chapter also brings to light the

power of the Java programming language in both its simplicity and its portability across all types of hardware and operating systems.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SOLVING THE PERSISTENT DATA PROBLEM

The SWIPNet application designed in Chapter IV performs well at delivering real-time Damage Control reports in a multicast environment. A major weakness however is the susceptibility to a network power outage. This chapter looks at different available storage models. It also looks at different ways to connect the storage model to the SWIPNet application to provide persistent storage. This chapter concludes by choosing the best model and creating an implementation to create a more robust SWIPNet application.

A. STORAGE MODELS

Different storage models are available to capture the state information that would allow the system to recover from a power outage. These include:

1. Flat Files

SWIPNet could dump reports and state information to a file that resides on the hard drive. In fact this was sometimes used in troubleshooting. However flat files do not provide a robust solution in a multi client environment because they are on the local machine. This prevents adequate sharing of the data. The data also has a tendency to get out of sync since it is recorded in several places. Also flat files provide virtually no security or integrity controls of the data placed in it. Based on this flat files are not a choice for implementation.

2. Relational Databases

A relational database is the defacto standard in data storage. Based upon a firm mathematical background and healthy history, relational database companies offer many solutions like Microsoft Access, Oracle, Sybase, SQL Server etc. Relational sharing of data is usually achieved through a client server approach. Data are retrieved and stored via a standardized language known as Structured Query Language (SQL). Relational databases also offer the required security and integrity constraints needed in a multi user

environment. Based on this a relational database was chosen as the storage model for implementation. Also two relational database were looked at, Microsoft Access 2000 and Oracle 8i. Oracle 8i provided a more robust Client Server database that provides for many clients in a Client/Server environment [Morrison00]. Oracle 8i provides many tools to a Database Administrator (DBA) to manage it. Microsoft Access 2000 is designed more for a single user [Balter99] and defaults the more robust client server handling to SQL server. Oracle 8i is the best choice, however based on funding and availability Microsoft Access was chosen as the database to implement.

3. Object Oriented Databases

A relatively newer approach for storing data, Object Databases store entire objects. Thus a Fire Object or Flooding Object could be stored and retrieved easier than a relational database. However this technology is relatively immature and there are not many vendors that offer this solution. Based on this Object oriented databases was not chosen as an implementation model.

4. eXtensible Markup Language (XML)

While Java is considered the portable language XML is considered portable data. Designed by the World Wide Web Consortium (W3C), XML has proven extremely useful because it provides an easy interchange of structured data across any platform, network or application. XML allows you to create a structure for your file known as a DTD (Document Type Definition) file. This file creates the structure and the data populates this structure in an xml file. Tags delimit pieces of data. These tags have no predefined meaning, they are solely defined in the DTD file. Thus new ML (Markup Languages) can be created like ChemML, BikeML, or DCML. XML provides a structure that can go to any level of complexity and there are tools, like XMLSpy, that can validate the correctness of your data. XML provides an excellent way to store and send data over the network and would work well within the SWIPNet design. Since a relational database was chosen for implementation an XML implementation will be saved as a recommendation for future work.

B. CONNECTION MODELS

1. Java Database Connectivity (JDBC)

Java provides JDBC Applications Programming Interface (API). This API provides code level access to SQL-based databases. Specifically this access would allow SWIPNet to connect to a central SWIPNet database from anywhere on the network. A DriverManager and Connection object [White99] manage this connection. The DriverManager controls and loads the driver for a particular database. The Connection object creates the connection to a database via Uniform Resource Locator (URL) and user name and password. Once a connection is made the Statement object in JDBC sends SQL statements to the database and returns the results to the user. The ResultSet holds these results. The ResultSet is then manipulated as necessary within the program. JDBC provides the link necessary to allow SWIPNet to provide a system level Initialization and to return the system to a consistent state after a power outage. This model is used in the design of the JDBCBridge object used in Chapter IV.

2. Server Side Models

Although not part of the SWIPNet design, Java offers J2EE (Java 2 Enterprise Edition) [Sun01]. J2EE offers several server side solutions for database connectivity, including Servlets, Java Server Pages (JSP's), and Enterprise JavaBeans. Server Side models process the desired query on the Server machine and return the result to the client. This solution works well with multiple databases and clients and is the right solution for an enterprise wide application. The use of this technology sounds promising but it is reserved as a recommendation for future study.

C. DC DESIGN REQUIREMENTS

As described before there are several options to integrate a persistent storage implementation into SWIPNet. The preferred method is to use JDBC-ODBC driver to allow provide two required features:

1. Initialization

During this phase a submarine crew keeps the database updated with current crewmembers and equipment. Also, technical diagrams and DC procedures would also need to be updated as revisions occur. During the start of the casualty the JDBS-ODBC driver would allow the database to initialize each sender and listener with the current names of the crew and other important data. This allows the sender to use a combo box to select the data to send vice typing it. This speeds the DC reports, which is vital in the casualty.

2. Power Outage Recovery

As the casualties are occurring, the reports that are sent are also dumped by the listener (or Server object) to the database. This maintains the current state of the casualty. If a power outage occurs then the data can be recovered and the system re-initialized using state data of the database and the Initialize.java and JDBCBridge.java designed in Chapter IV. This assumes that battery backups were either not used or have depleted their reserve.

D. PERSISTENCE DESIGN USING MICROSOFT ACCESS

The Entity Relationship model in Figure 26 shows the entities and relationships necessary to capture DC state information. The entities also have attributes that are seen in Figure 27. A total of nine entities, shown as rectangles, are needed. Each entity represents a specific table of information that will hold DC information within the database. The relationships between each entity are also shown in appendix A create the desired Integrity Constraints within the database. These constraints are based on the Business rules described below. All relationships are One to Many with the exception of Location relationship which is Many to Many and creates the required extra table.

1. Entity Description

- ❑ *Casualty* – This object captures the type of casualty (Fire, Flooding, etc). It also holds the Time the casualty started and the specific casualties Status. Each casualty would also have a Man In Charge (MIC)
- ❑ *Crew* – This object captures all crewmembers currently on board.
- ❑ *Compartments* – This object captures every compartment onboard a submarine that a casualty could occur in. For example ERUL – Engine Room Upper Level, AMR2UL – Auxiliary Machinery Room 2 Upper Level, or FCML – Forward Compartment Upper Level.
- ❑ *Reports* – This object captures all reports that are sent to and from Damage Control Central. These reports capture the source, time sent and the specific messages that are needed to control the casualty. They would represent the old way of using the sound powered phone circuit.
- ❑ *Sensors* – Captures all sensor information that is monitored by the Virtual Damage Control system. Ex: DP-023 a depth sensor needed to monitor the ships current depth.
- ❑ *Atmospheres* – The atmosphere captures the current level of atmospheres (e.g. O2 level) and shows if that limit is currently being met. This information is important for determining if breathing devices are needed by crewmembers.
- ❑ *Rigs* – This object captures the compartment rigging status of key components throughout the ship. Rigs place the ship in maximum sustainability posture during casualties. These rigs are performed by crewmembers and reported. The necessary information needed to be captured is the Type (e.g. Rig for Fire), the Status (Rigged or Not) and the Location (e.g. ERUL – Engine Room Upper Level)

- ❑ *Hoses* – Deployment of hoses are used in many casualties. Hoses are labeled by the compartment they come from and is captured as it's hose Types. Teams of crewmembers usually man one hose.
- ❑ *Breath_Device* – They are different types of breathing devices onboard submarines (e.g. OBA – Oxygen Breathing Apparatus, Mark-V – Gas Masks) and knowing which devices is being worn by what crew member is important because of the time restrictions and atmosphere restriction of each device.

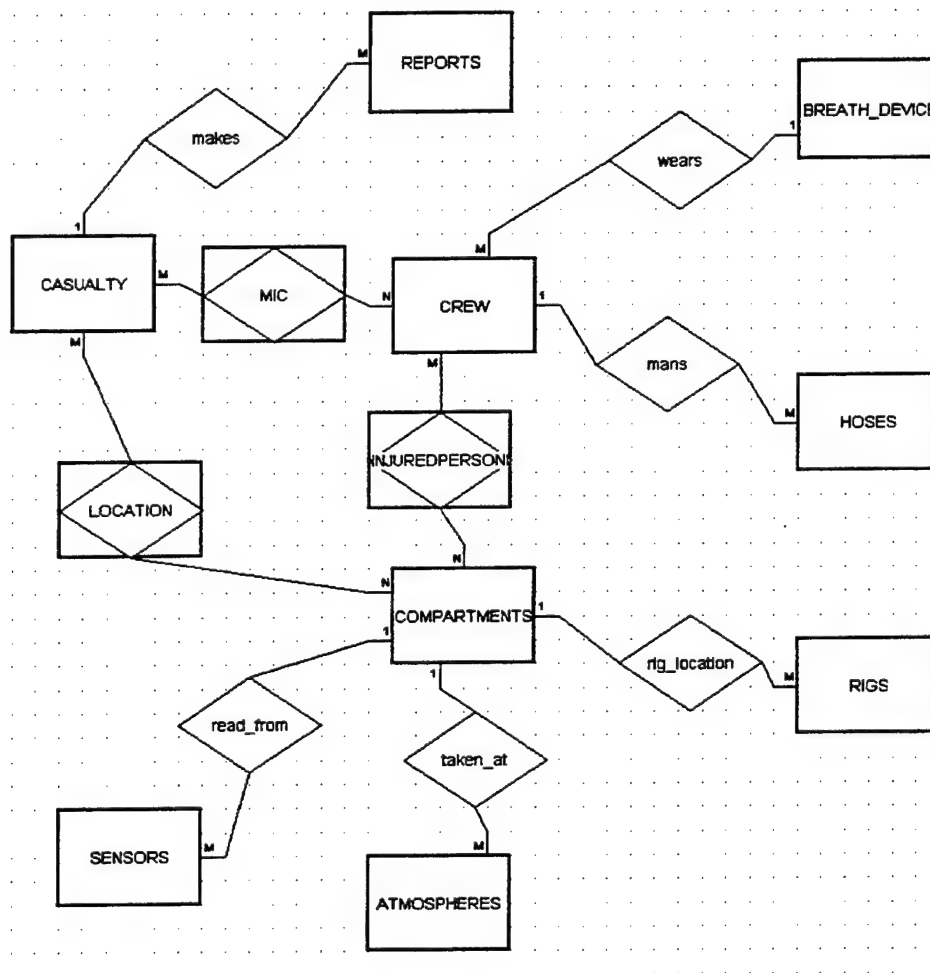


Figure 26 - Entity Relationship (ER) Diagram

2. Relationships

The following describes the desired relationships needed in the SWIPNet DC design. Basically they are the business rules and constraints between each entity (See Figure 26).

- ❑ **Injured Person** - Only one Crewmember can be injured in many (since can transfer across) Compartments, and a Compartment can have many injured Crewmembers.
- ❑ **Man in charge (MIC)** - Many Crew MIC (on different levels) per Casualty, and many Casualties for each Crew MIC.
- ❑ **Wears** - Breath_Device are issued to many Crewmembers and a Crew member can have only one Breath_Device.
- ❑ **Mans** - Many Crew members allowed per Hose, but only one Hose to each Crewmember.
- ❑ **Location** - One Casualty can occur in many Compartments and one Compartment can hold many Casualties.
- ❑ **Read_From** - Many Sensors per Compartment, but one Compartment per Sensor.
- ❑ **Taken_At** - Many Atmospheres per Compartment, but only one Atmosphere type per Compartment.
- ❑ **Rig_Location** - Many Rigs per Compartment, but only One Compartment per Rig.

- ❑ **Makes** - Casualties can have many reports, but only one report per casualty type.

3. Microsoft Access Implementation

This is the actual implementation in MS access. Figure 27 shows this implementations entity and corresponding attributes. Each entity would represent a table in the database and the attribute would represent a column. A relationship only appears as a table if it is a many-to-many relationship.

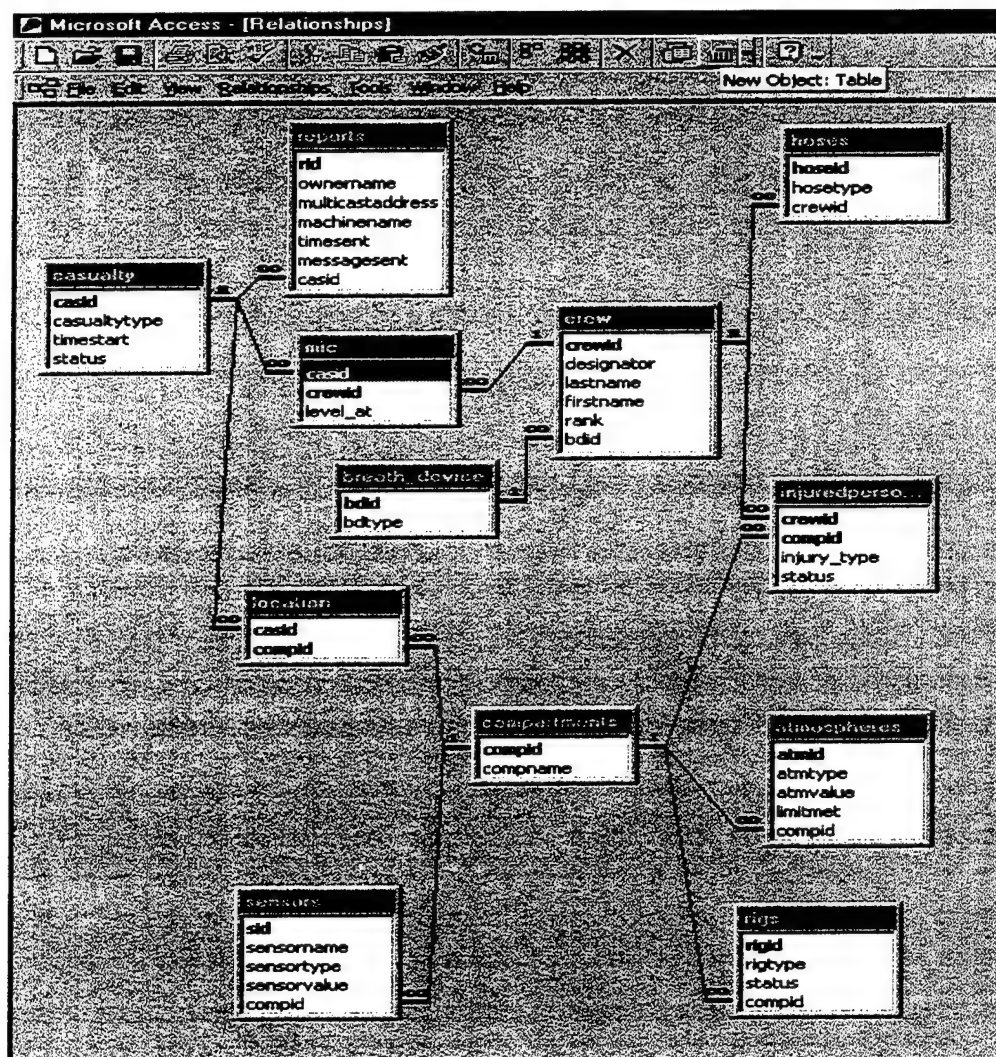


Figure 27 - Relationship Schema/Relationships and Attributes

4. Connecting the Database

The actual implementation of this connection will become a recommendation for future work. It should be noted that currently SWIPNet has been designed to easily allow this implementation to occur by creation of the Initialize and JDBCBridge objects, see Chapter IV, that can be sent just as easily as a Fire class, Flooding Class, etc that are currently being sent.

E. DATA INTEGRITY FEATURES USING MICROSOFT ACCESS

The following section describes features that would be used to provide data integrity during a casualty as state data is updated to a database. Since this database would be used to provide initialization data to DC consoles at the start of the casualty it is important that the data is not tampered with before a casualty and that the database does not become corrupt during the casualty. It might also be important to use this data after the casualty to identify a problem or help the crew train better during drills. Areas that are addressed are security, database concurrency locking, data schema integrity, and backup [Balter99].

1. Security

a. Encryption

This makes the data in the database indecipherable from data in a word processor, disk utility, etc. You can get to it by Tools|Security| Encrypt/Decrypt Database

b. User and Group Accounts

These options under Tools|Security allows you to create groups and assign permissions of different objects, such as tables, queries, etc. much like the security model in WindowsNT itself. For example, a Group such as "Data Access" is given permission to all forms to enter data. The DBA could then add a user to that group and then they could access the forms and enter data.

c. DB Password

Microsoft (MS) Access 2000 allows you to set a password for the entire database.

d. Security Wizard

Like most MS Office products, wizards are there to assist in complex processes. Security in Access is no exception. The Security Wizards allows things like: 1) Designating objects that need security, 2) Creating users, 3) Setting passwords, etc.

MS Access 2000 also allows a new feature called an MDE file, which is a copy of the existing database with all the underlying source code removed, thus protecting the most of the efforts of the designer.

Most security aspects are applied to the Database and the objects in the database. MS Access also allows linking tables to other backend sources like SQL Server. This client/server approach also has built in security feature similar to some described previously.

2. Protective Locking of Data in a Multiple User Environment

The following features provide different levels of granularity for locking the data. It is recommended that Table and Recordset locking be used for the SWIPNet DC design.

a. Record Locking

Only the record the user is editing is locked

b. Page Locking

A 4KB page with the record being edited is locked

c. Table and Recordset Locking

The entire table or recordset with the record being edited is locked.

d. Opening an Entire Database with Exclusive Access

The entire database is locked, unless that user has opened the database in Read Only mode. If so, then others can open the Database in Read Only

Tools|Options|Advanced sets the above global multi-user settings for the user. No Locks (optimistic approach), All Records or Edited Records (pessimistic approach) are the choices that choose these settings. Within each query, form or record the Properties sections also allow specific locks to be applied specifically to those objects during development. Reports don't offer locking choices because they cannot be modified. No Locks choice means the page of data will not be locked until Access tries to write the changed data to disk and is considered the least restrictive. All Records is the most restrictive, and other users can only view the data. Edited Records takes a 4KB's surrounding the record is locked.

3. Data Integrity

Referential Integrity is an option on one-to-many relationships in the database. This concept is used within the database created for this project. It enforces a set of rules to help prevent orphan child data within the database. Cascade Update and Cascade Delete are also used in this database and work similarly as described in class.

Menus are developed in this database to guide the user. This Main Menu is presented at startup and the underneath database is hidden from the user. This also helps maintain data integrity by minimizing the confusion for the user.

4. DB Recovery and Backup

MS Access allows creation of a working backup through what it calls a "Replica". This Replica is a linked copy of your existing database, designated the Design Master. Once a Replica is created from the Tools|Replication menu, changes in the data can occur by a user in the Replica or the Design Master. Any structural changes, however, can only be applied in the Design Master. The Periodic Synchronize Now option allows the Design Master to update the Replica (ie. Backup). Synchronizing can create conflicts between the databases. Access has an option to resolve the conflicts,

that allows you to keep existing data, keep revised data, overwrite with conflicting data and overwrite with revised data. A Replication Manager is also available to automate a lot of these processes. Synchronizations can also be scheduled during different times of the day. Partial replicas are also allowed (i.e. a subset of the database).

WindowsNT 4.0 and 2000 [Minasi00] also has some backup features: the Backup utility allows periodic backup to a tape drive to help protect the data. Also Windows NT allows two fault tolerant schemes Disk Mirroring (RAID1) and Striped Sets with Parity (RAID5). RAID stands for Redundant Array of Independent Disk. These protection options help protect data in case of a disk failure. RAID1 actually creates, transparent to the user, a second disk image in case the first disk fails. RAID5 places parity stripe information on all disk. If one of the disks in the set fails then that disk can be replaced and the data regenerated to recover all data.

F. CHAPTER SUMMARY

Chapter V looks at different storage models that could be used as a persistent data model for the SWIPNet application developed in Chapter IV. This model was developed using Microsoft Access 2000. A recommendation for future work is to use the JDBCBridge class and the Initializer class developed in Chapter IV to provide connectivity to this database to help overcome the design weakness that SWIPNet is vulnerable to, i.e. network power outage. This connectivity should also provide for a Initialization sequence for the system that can be designed for the specific boats who will use SWIPNet.

VI. SUBMARINE NETWORK ANALYSIS

The focus of this chapter is to provide some insight into the integration of a virtual Damage Control (DC) wireless component application into a submarine type LAN. The LAN that has been picked for this analysis is a Non Tactical Data Processing System (NTDPS) currently being considered for the Virginia class submarine. A diagram that has been proposed for this system has been obtained with permission of NAVSEA and has been used to develop the test model used in this chapter. This diagram is called the Original Network and is shown in Figure 29. The tool used to build and analyze this network is OPNET Modeler 7.0B.

A. GOALS

- ☐ What applications should the wireless clients support and is the network robust enough to handle them?
- ☐ Prove that a submarine DC application can run on a wireless network assuming all other network loads are present.
- ☐ Describe the network/traffic patterns in a normal 24 hr day and identify the peak traffic times.
- ☐ Determine the maximum number of wired and wireless workstations the network can support?
- ☐ Define and test the Longest Path scenario.
- ☐ Identify the average and Longest Path Ethernet delay, average throughput and average utilization in terms of number of clients, types of applications and distance to servers?
- ☐ Validate the simulation (OPNET model) with an analytical model.
- ☐ Describe the redundancy and reliability of the Network. Identify weaknesses and recommend improvements.

B. DEVELOPMENT PLAN

To meet the above goals a network had to be designed from diagrams received from NAVSEA code 450. The flowchart, Figure 28, outlines the major steps that were used to develop this model. This flowchart outlines all the major steps of this chapter concluding with a Chapter Summary.

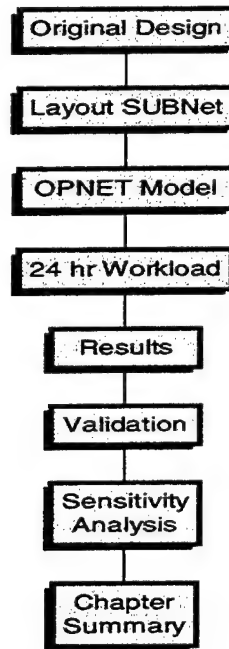


Figure 28 - Flowchart of Network Design

The Original Design was requested and obtained by NAVSEA code 450 is shown in Figure 29. It was the basis for the models develop and was modified under an assumption set used to develop the Modified LAN. Once a modified diagram, Figure 30, was developed, the next step was laying the components out, as they would appear on a submarine. This submarine modeled assumes a 250 ft long by 34 ft wide (living area). These dimensions needed to be set to allow OPNET modeler to accurately calculate the results of this model. Once the components were laid out, the OPNET model was built and a 24 hr workload was applied to obtain the results. The submarine DC application is part of this 24 hr workload. Validation of the OPNET model was accomplished by comparing a "Longest Path" scenario's Total Delay to a calculated

delay value described fully in [Sadiku95]. Sensitivity analysis attempts to stress the system to determine the maximum number of clients and workload that could be applied to this system. This analysis concentrates mostly on the wireless part of this system. The conclusions summarize the strengths and weakness of the designed system and makes recommendations for its implementation.

C. MODIFYING THE ORIGINAL DESIGN

The system designed was obtained from the Naval Sea Systems (NAVSEA) command (Figure 29). The proposal is founded on ideas presented from Electric Boat.

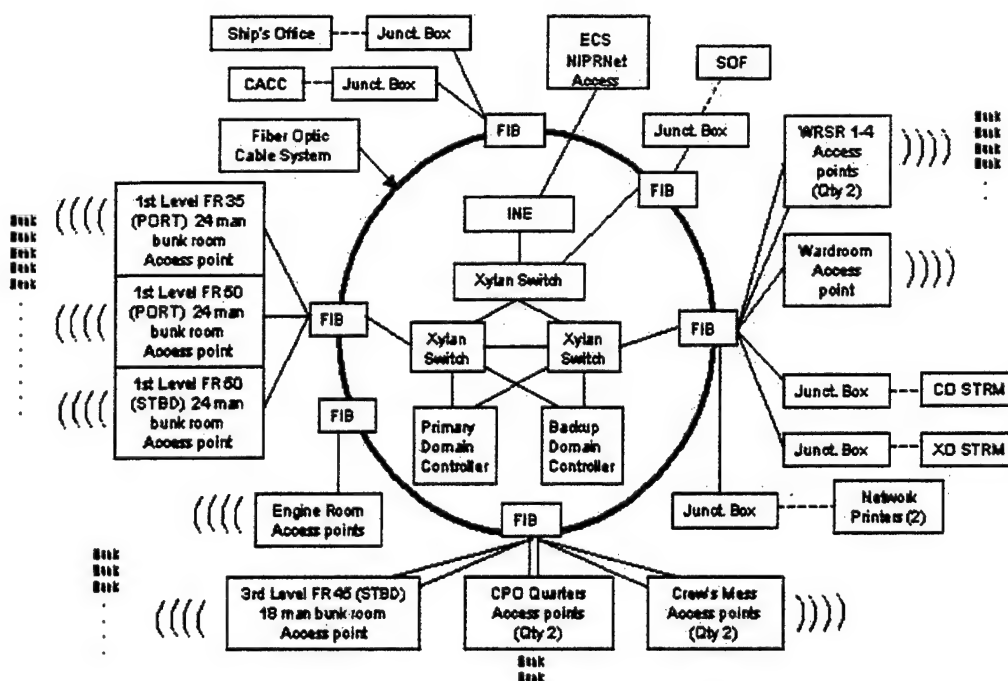


Figure 29 - "Original Design" LAN

Per phone conversations with NAVSEA, the design was modified to adhere and comply with the OPNET Modeling tool. Figure 30 depicts the modified design. It should be noted that this layout and all further steps were done without any further inside knowledge of the NTDPs system. This was done to ensure the confidentiality of the actual system.

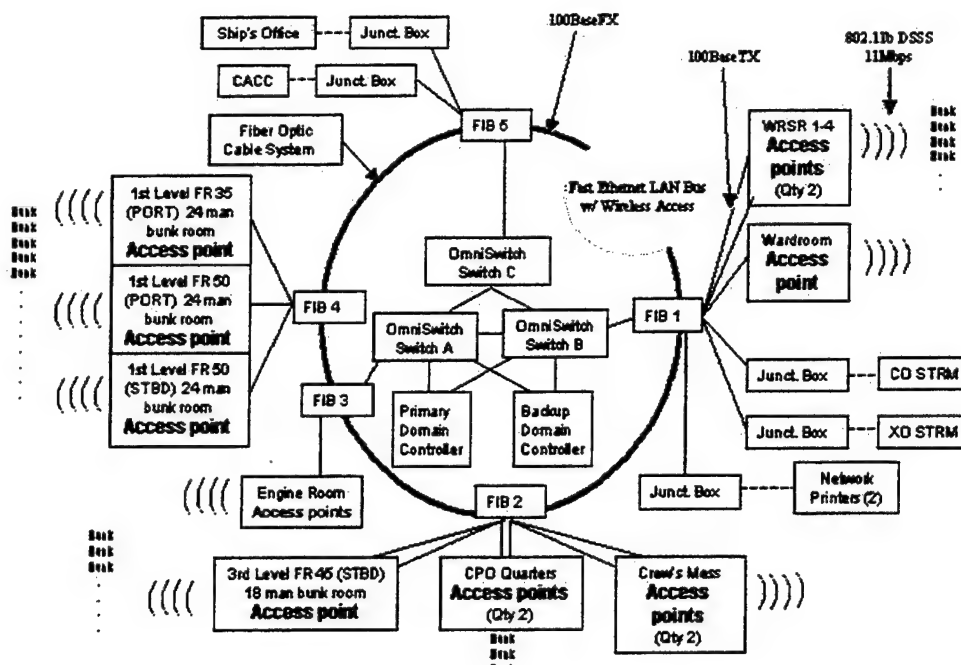


Figure 30 - "Modified" LAN

This diagram is used to layout Components shown in Figure 31. It is important to note that in order to map the original design a series of assumptions were made which are listed below.

D. ASSUMPTIONS OF MODIFIED LAN:

1. LAN Classification

The original design appears to utilize FDDI or SONET over ATM, operating over an optical fiber. However, the Original Design is actually a 100 Mbps Fast Ethernet BUS LAN with 100BaseFX and 100BaseTX, integrated with wireless Access Points and Clients throughout the submarine.

2. Link Classification

The Cable Plant (wired transmission medium) on the NTDPS in actuality is a 100BaseFX (Fiber). This portion of the network was modeled as 100BaseTX (Twisted Pair) for two reasons:

For one, the rated bandwidths are the same throughout the wired portions of the network (i.e., 100Mbps), so implementing it with fiber would provide no significant advantages with regard to bandwidth. However, it should be understood that the advantages of fiber are its immunity to EMI and electrical interference. In addition, fiber has a Lower Signal Power loss, provides better security, is lighter and will not produce fires or explosion in comparison to twisted pair media [Quinn97]. Clearly these are all important factors for designing a LAN within the confines of a submarine. However, the focus of the study is delay, utilization and throughput and no significant change in the conclusions using 100BaseFX with the distances involved are foreseen in this study.

Another primary reason twisted pair was selected is that the OPNET Modeler does not provide a default 100BaseFX component. One can custom design a 100BaseFX component by mapping 1000BaseFX and changing some internal variables, but the focus was on designing the overall network. Attempts to convert the variables were too time consuming and was not assured the design would truly reflect the correct parameters of a 100BaseFX component.

3. Fiber Interface Boxes (FIB) to Switch Conversion

The initial diagram shows certain Access Points (AP), which connect to a FIB. As discussed above, the simplified cable plant uses 100BaseTX Ethernet Bus, thus the FIBs are essentially just switches and no longer need to convert fiber signals to electrical signals and vice versa.

4. Grouping of Traffic into Subnets

The proposed LAN shows that certain FIBs have classes of traffic assigned to them. For example, the officer's staterooms and wardroom access points are connected to the same FIB (switch). This general grouping of access points was kept in the design,

and modeled as Subnets. All Subnet are chosen based on their respective traffic stream (e.g., officer's message traffic requires the Gold subnet, wireless clients, and access points to be located in the wardroom and stateroom areas).

5. Placement of Components

The physical locations of compartment components (Servers, Switches, Access Points, and Clients) were selected based on what made the most sense. They are assumed to be located in the middle of the compartment.

6. Selection of Backbone Switches

While trying to obtain configuration data about the Xylan switches, it was discovered that Alcatel, Inc. has purchased Xylan, Inc. This made obtaining data for use in the OPNET model nearly impossible. Based on this, the choice was made to use a comparable switch from Alcatel. This selection was the Omni-Switch, ESM-100C-32W-2C, 32-port 10/100 Fast Ethernet backbone module [Products00]. However, after phone conversations with Alcatel, it was discovered that there are no OPNET Model libraries developed for these switches (note: they are looking at it for the future). Because of this, a comparable Cisco router was used to model the backbone switches within the OPNET Model.

7. Others

For simplicity, the Special Operations Force (SOF) junction boxes and NIPRNET/INE connections, shown in Figure 29, were not modeled. In addition, it is assumed the primary and backup domain controllers (PDC and BDC) have one NIC card per server. These assumptions were made to concentrate the focus on the backbone and wireless portions of the LAN.

E. LAYOUT OF THE SUBMARINE NETWORK

1. Physical Layout

a. Submarine Cross Section Description:

- Dimensions: 250 ft long and 34 ft in diameter
- Five Subnets: Red (Engine room), Blue (Enlisted Berthing), Green (CPO Berthing), Gold (Officer Berthing) and Orange (Command and Control Center)
- Control Center (Figure 33) is located in Navigation (NAV)
- Junction Boxes: are Ethernet drops for wired workstations and printers

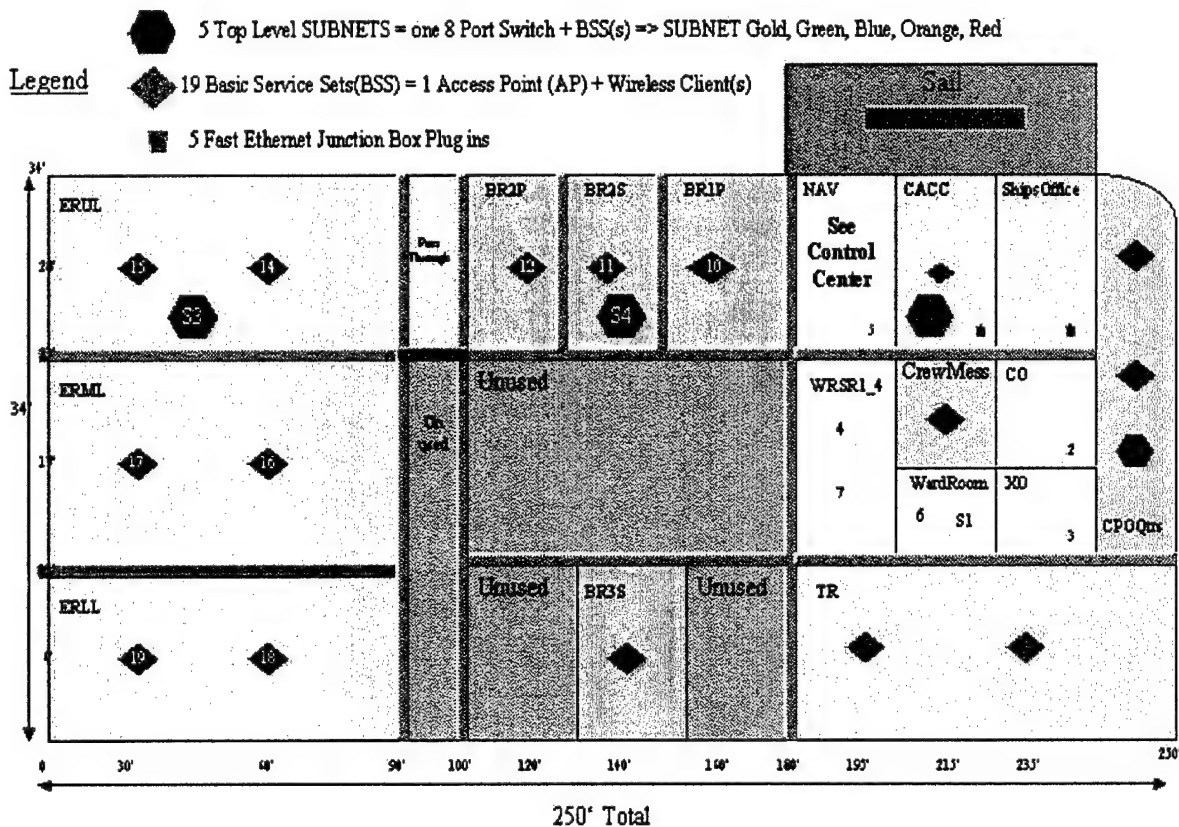


Figure 31 - "Modified" LAN Physical Layout

b. Control Center and Link Layout Description

- ❑ Server to Access Points Links: Fast Ethernet, CSMA/CD protocol with 100BaseTX [Hammond86]
- ❑ Access point to wireless client link: uses the 11MBps, IEEE 802.11b, CSMA/CA, Direct Sequence (DS/SS) protocol [Tanenbaum96].
- ❑ Transport and Network Protocol: TCP/IP throughout the entire network.
- ❑ Subnet: One eight port Ethernet switch, multiple access points and multiple wireless (SEE OPNET RESTRICTION).
- ❑ Control Center (Figure 32): Two NT 2000 Servers and three backbone switches (A, B, & C).
- ❑ OPNET RESTRICTION: OPNET allows one AP per subnet with multiple clients. This is known as a Basic Service Set [Guide00]. Therefore, the Gold, Green, Red, Blue, and Orange subnets themselves contain Subnets to house each of the AP's shown. For example, the Gold subnet contains 3 subnets (called AP4, AP6, AP7) that house one AP and their respective wireless clients.

- Backbone Switches (A, B, & C) - Cisco C5000 Switches are used because they provide the most compatible with the Alcatel Omni-Switch. These switches have 24 Ethernet and 12 Fast Ethernet ports.
- Subnet Switches (within Subnets) - These are basic switches with 8 Fast Ethernet ports. They are equivalent to the FIB discussed earlier.
- Access Points (AP) - Wireless routers that route traffic to multiple wireless clients. They are configured in Infrastructure mode (vice ADHOC or Peer-to-Peer) and set for 11Mbps, Direct Sequence routing.
- Wireless Clients - Windows 2000, 11Mbps, Direct Sequence. These clients are assumed stationary and can run all required workloads.
- Junction Boxes - Provide Ethernet plug-in access to regular NT workstations and network printers.

F. KEY SERVICES AND MEASURED VALUES USED IN STUDY

1. Key Services

The NTDPS system will serve the crew of the submarine by providing certain key wireless and wired functions. The system level services include:

a. Class Roaming Service

Providing wireless Windows Server access to five classes of crewmembers. These classes are Enlisted, Chief Petty Officers, Officers, Engine Room Watch Standers and Control & Command Center Watch Standers. The respective subnets will only allow wireless access when their wireless clients are within the area of their subnet. This "class type" isolation is done for three primary reasons. For one, they provide a separation of traffic and better security. Next, they help to ensure that one subnet will not become overloaded by a non-domain user (e.g., 50 enlisted members

overwhelming the Gold - officer's subnet). Lastly, since the system is designed for non-tactical data (usually personal), it provides the required access within the most likely place that the respective user would be when trying to access the network (e.g., enlisted crew within the berthing area, Chiefs in the CPO quarters, etc).

Class roaming service could be avoided if it is determined that roaming capability of one class through all domains of the wireless clients becomes more important than the above three factors.

b. Application Service

The Windows Server and wireless clients will be able to perform a range of Client/Server applications: HTTP, FTP, e-mail, Telnet, Database Queries, and Streaming Video and Audio.

c. Plug-in and Printing Service

There are also several Windows Workstations and network printers that are able to plug into the hardwired section via junction boxes; printing is also a desired service.

2. Metrics Studied during Simulation

All metrics were analyzed, however only key metrics are presented in this chapter.

a. Packet Based Metrics (TCP/IP Network layer)

- Mean Ethernet Delay (seconds)
- Mean Subnet-to-Subnet Throughput (bps)
- Link Utilization
- Total Arrival Rates (packets/sec)
- Number of hops
- Distance between hops (ft)
- Queuing Delay (seconds)

b. Application Metrics (TCP/IP Application layer)

- Mean Inter-repetition times (seconds) - Poisson distribution
- Mean size of application transfer (Bytes)
- Mean response time for each application (seconds)
- Type of application (HTTP, FTP, Email, Stream A/V, Database)

3. Parameters Defined for Simulation

The following are parameters that were studied that could affect the performance of the system. They are listed in two primary areas, System and Workload. Again they are listed for completeness but only key parameters will be discussed later.

a. System Parameters

- Bus speed (Mbps)
- CPU speed of server
- CPU speed of wireless client
- IP forwarding rate (bits per second)
- Buffer size (bits)
- Packet length (Bytes)
- Types of links (100BaseTX, Wireless)
- Number of subnets
- Background utilization of the links (percent)

b. Workload Parameters

- Type of application service (HTTP, FTP, DB, Email, Stream A/V)
- Frequency and file size (Bytes) workloads of the applications
- Workload per time of day
- Location of components in the submarine
- Distance (feet) of server to clients
- Number of wireless clients
- Hops between server and client
- Number of subnets

4. Factors Varied during Simulation

The following are factors that were changed during the OPNET simulation.

- ☐ Arrival rates (packets/sec)
- ☐ Application each client runs (HTTP, FTP, EMAIL, Streaming A/V, Database)
- ☐ Distance (feet) and hops between server and clients
- ☐ Inter-repetition times (seconds) of applications
- ☐ Workload and number of clients

G. OPNET MODEL

The modified LAN was built using OPNET Modeler version 7.0B. As noted earlier, there are five subnets in the design. Located within each subnet are a switch, access point and wireless client. The Gold Subnet also contains the PDC and BDC, Cisco backbone routers and serves as the central brain of the network. The OPNET Model is shown in Figure 33.

1. OPNET Model Assumptions

There is only one NIC card per server, therefore only one connection is allowed between the Cisco backbone routers (Switch A, B, C). Windows 2000 servers support all desired applications; HTTP, FTP, e-mail, Database, and Streaming A/V.

A Basic Service Set (BSS) constitutes a subnet with a respective Access Point and Wireless client. OPNET allows only one access point per subnet [Guide00]; therefore the original assumption of having multiple access points (as illustrated in Figure 32) has been simplified to one.

The settings within OPNET for servers, switches, access points and wireless clients to be default, unless specifically mentioned otherwise.

It is assumed all applications perform under the pretense of the "Best Effort Service Model" and use TCP/IP over Ethernet to route packets.

2. Problems Encountered

A buffer overflow led to dropped packets on the MAC layers of the access points and wireless clients. In an attempt to resolve this problem, the buffer size was increased and IP forwarding rate of both the access points and the wireless clients by a factor of four and ten, respectively. The resulting buffer size of the access points and wireless clients are 1,024,000 bits each. The IP-Forwarding rate of the access points is set to 50,000 bits/sec and the wireless clients to 5000 bits/sec. Under these conditions I were able to simulate HTTP43KB, HTTP4KB, FTP1MB, FTP300KB, EMAIL40KB, and Database 32KB successfully on each client. Increasing these values any higher gave no added benefit. That is no further load could be added within a successful simulation.

Start time offset for the applications must be at least 15 seconds to allow the routing information protocol (RIP) to run. If it is set to zero, an OPNET compile error occurs. Also, it was not possible to achieve 100 % utilization for throughput on wireless/wired system. It can only be concluded that either the OPNET program is

flawed or the bottlenecks in the system were not readily apparent. Therefore, most conclusions are based on mean response time.

3. OPNET Diagram

Figure 33 shows screen shots of the actual OPNET model. The top level contains five subnets described earlier. The Gold subnet contains the control center and the respective BSS subnets. Inside AP4, AP6, and AP7 are the AP and the number of wireless clients needed to run the various simulations.

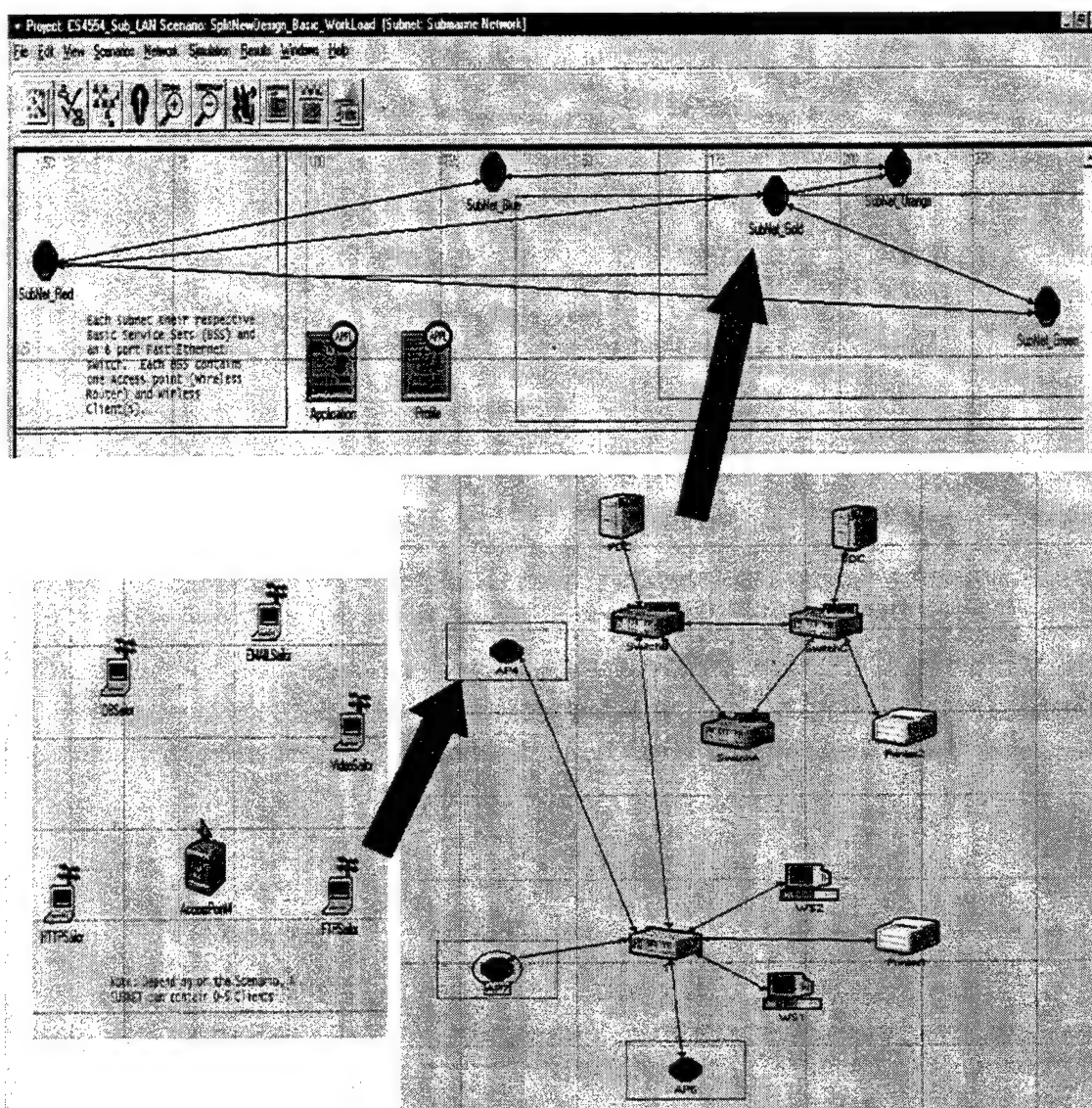


Figure 33 - OPNET Model (with expanded Gold Subnet)

4. Configuration of the Simulation Components

The Configuration Panels are included to aid in future development with OPNET modeler. This program is very powerful yet its complexity takes many hours to learn. Showing the configuration panel shows the default values used during the simulations and the values that were changed to create the desired model.

a. Wireless Client (Example of HTTP client)

The default values shown in Figure 34 that were changed include the Data Rate (bps), Physical Characteristics, Buffer Size (bits), IP Forwarding Rate (bits/sec), and this case a HTTP workload was applied to the Supported Profiles.

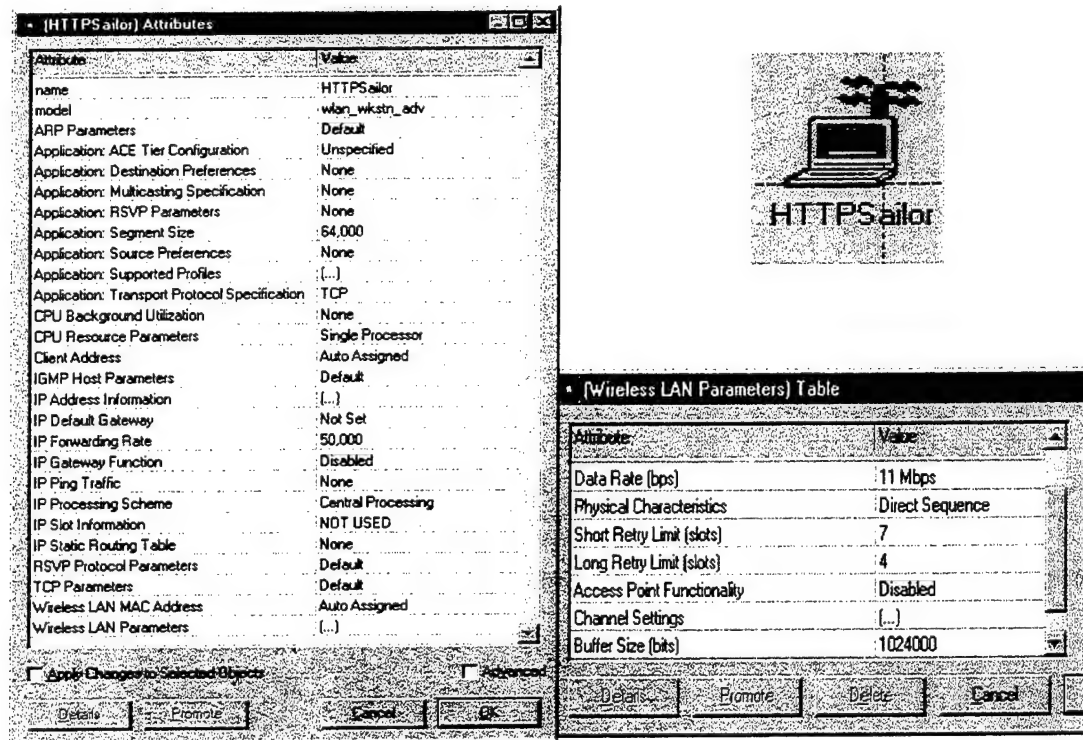


Figure 34 - Example Wireless Client Configuration

b. Access Point (Example of AccessPoint4)

The default values shown in Figure 35 that were changed include Data Rate (bps), Physical Characteristics, IP Forwarding Rate (bps), and Access Point Functionality.

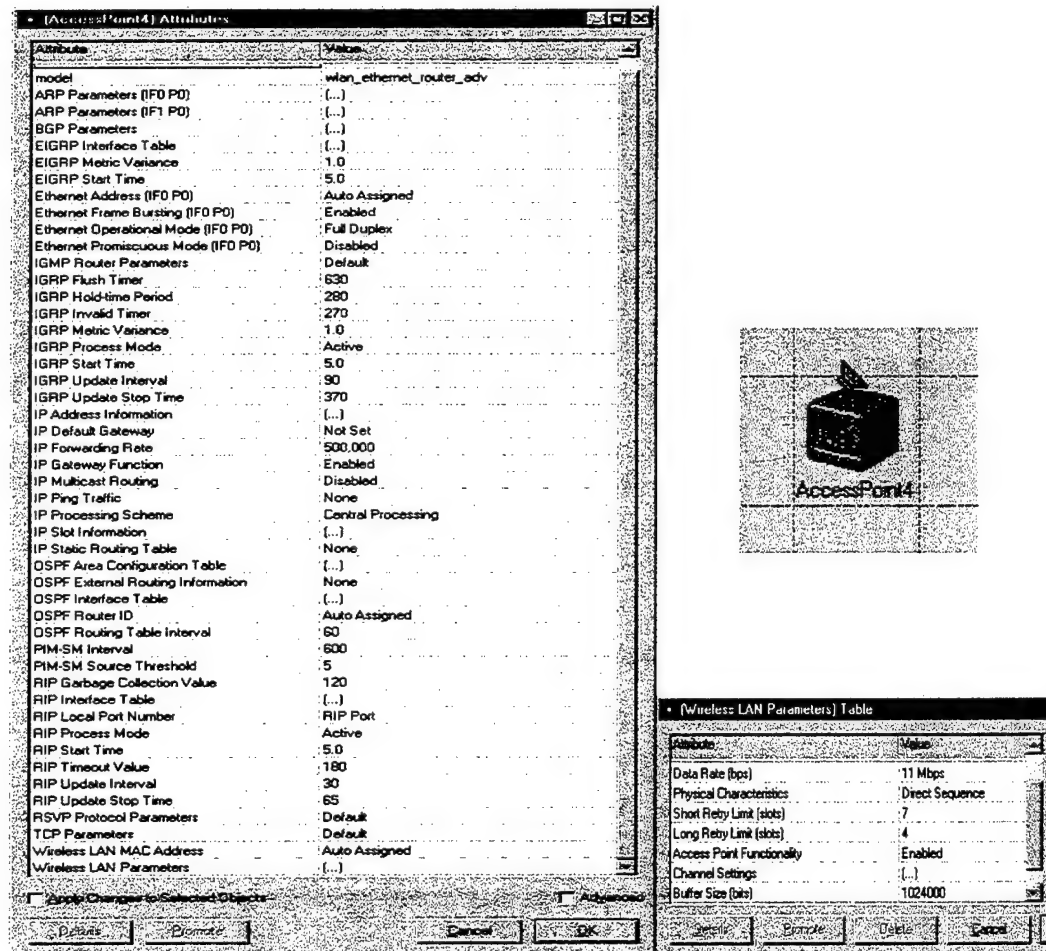


Figure 35 - Example Access Point Configuration

c. Domain Controller (Primary Domain Controller)

CPU Resource Parameters, as shown in Figure 36, were changed to support three processors for the PDC with the BDC disabled. This allows a probe to

analyze the full traffic stream between switch B and the PDC. Three processors were needed to ensure the PDC was not the limiting node in the network under these conditions.

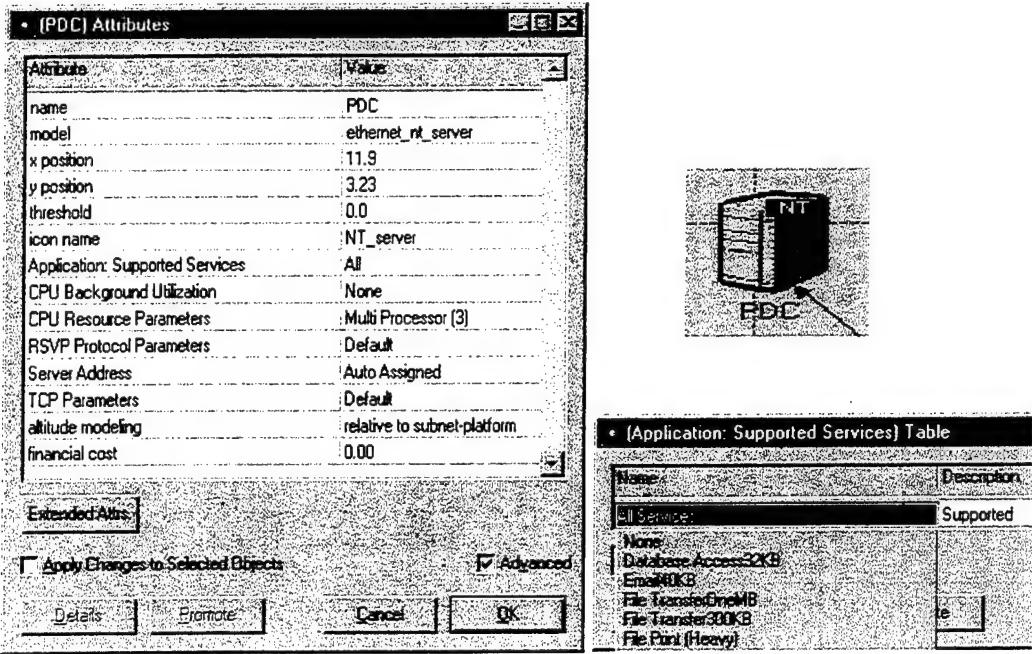


Figure 36 - Example Domain Controller Configuration

d. Workload Profile (All profiles developed)

The profiles, as shown in Figure 37, were created to hold the applications that will run on each of the clients. HTTP profile contains HTTP43KB and HTTP6KB. FTP holds the FTP1MB and FTP300KB. EMAIL and DB holds the EMAIL40KB and DB32KB respectively. Video holds the 128X120 video stream application. WorkloadOne contains all applications and the PrintColor holds a print application, but both were unused.

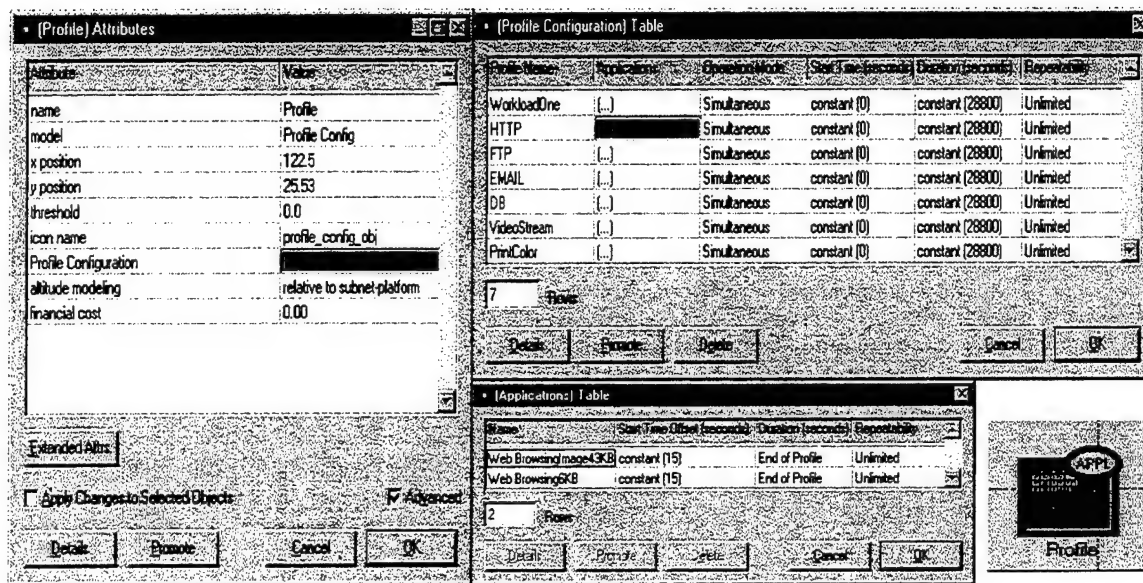


Figure 37 - Profile Configuration

e. Application (Example of HTTP6KB)

Applications are configured as shown in Figure 38. The Inter-arrival Time is the time between each HTTP request. The 6KB is based on a 1000 bit ASCII page and 5 objects per page that average 1000 bits. The other workloads were similar calculated and explained in detail in the next section.

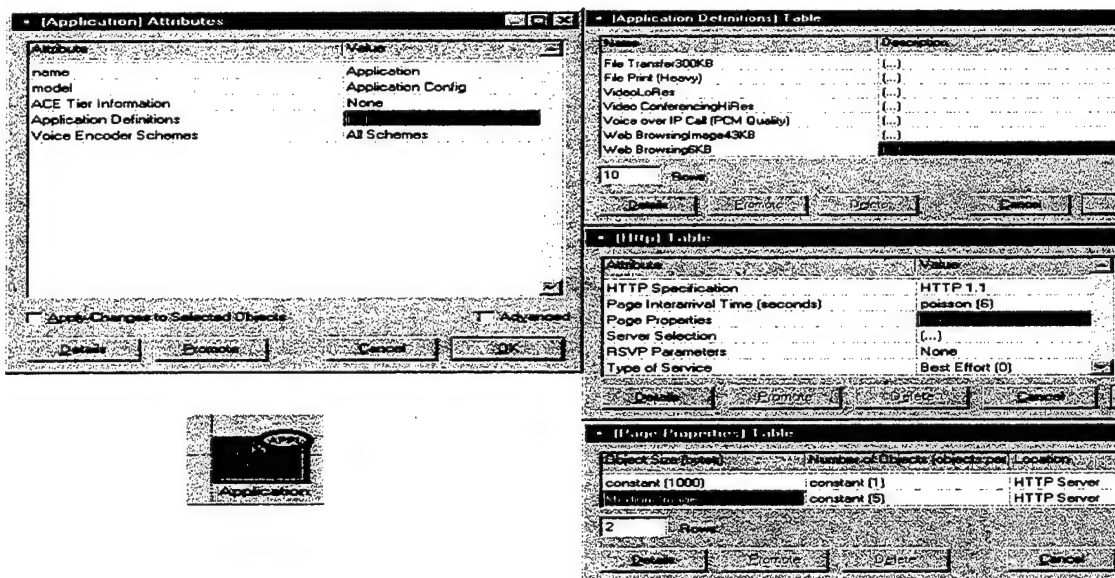


Figure 38 - Example Application Configuration

5. Other Components

The following components in Figure 39 were included in this study but their default values not manipulated. Also, No workloads were applied to the printers or workstations.



Figure 39 - Other Modeled Components

H. WORKLOADS AND TASKS DESIGN

The purpose of this section is to identify basic Submarine network task, the “time” peak loading occurs during a normal workday and the applications needed to support the basic task list onboard a submarine. Figure 40 shows a flow chart used to determine this data.

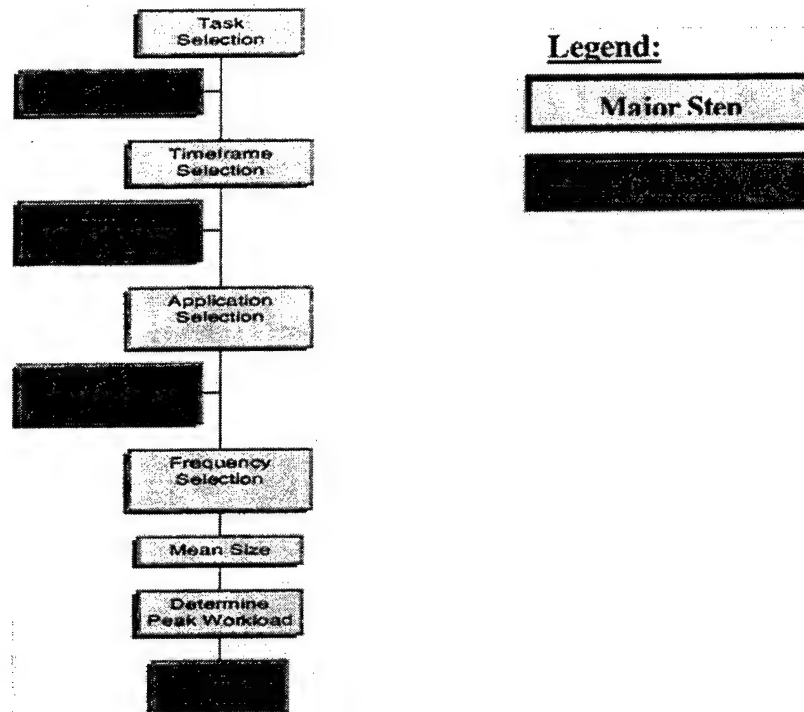


Figure 40 - Workload Selection Flow Chart

15 task areas were identified that could be running on the modified NTDPS LAN during different times of the day.

1. Submarine DC Drills

Represents SWIPNet Communication flow and reporting during daily DC drills.

2. Log Taking

Log taking includes the recording and analysis of equipment parameters by watch standers throughout the ship 24 hours per day.

3. Repair Maintenance Management

Includes maintenance communications and approval tracking after the maintenance has been completed.

4. Supply Inventory

Would include the network load for an application that purchases parts, tracks current supply, and updates inventory.

5. Preventative Maintenance

Schedule Preventive Maintenance (PM's) and track their completion.

6. Message Routing

Consists of message routing through the Chain of Command. Those messages would include message retrieved from the daily broadcast and messages used to communicate to the crew about daily routines.

7. Watch Bill Scheduling

This includes the load that would route inputs and changes for a dynamic watch bill for ship's watch standers. Would also include the crew downloading this watch bill as needed to determine what watch they have.

8. Refit Planning

Includes the creation and distribution of a refit plan. This would only occur near the end of a patrol before entering a refit period.

9. Plan of the Day (POD) Deployment

Includes the POD creation and distribution to the crew.

10. Qualification System

This load tracks the qualification of all crewmembers. Specifically, Smart cards verification, digital signatures and tracking point counts contribute to this loading.

11. Fitness Reports (FITREP) and Evaluations (EVAL) Tracking

Officer / Enlisted performance would load the system by taking inputs from crew members and tracking the writing and approval of the report by their superior.

12. General Record Storage and Retrieval

This would be digitized services that allows viewing and updating of medical, dental and pay records.

13. Online Training

This service is an automation of training that includes digitized lesson plan retrieval, test and training tracking and training scheduling.

14. Online Ship Inspection and Exam History

This load is dedicated to the operational exams (e.g. ORSE, TRE, NTPD) that occur each patrol. Specifically, it would include lessons learned distribution and deficiencies correction.

15. Crew Leisure Activities

Considered the heaviest network load but assumed to only occur in the evening and at night. This load includes online training, digitized movie/sound retrieval, web browsing and email, and general network entertainment needs. The task areas are grouped into specific timeframes to simplify the calculations. Six timeframes were identified as unique (see Figure 42). Then five applications (depicted at the top of Figure 41) were identified to support these task areas. Next, the frequency at which these applications would occur within their respective timeframes was determined. With this data in-hand, a graph was created to illustrate that peak loading would occur during the

timeframe of 1900–2100. These two hours helped to scope the workloads applied to the OPNET model. It is important to note that these workloads have not been validated. It is recommended that real usage patterns be obtained by attaching a monitored system to a real submarine's LAN system.

I. WORKLOAD MATRIX AND TIMEFRAMES

1. Matrix

The numbers should be read as follows: “Mean application Frequency” for given timeframe/ “Mean Size” of upload or download/ “Timeframe #” the event occurs.

Task Area	HTTP	FTP	Stream A/V	EMAIL	Database
Submarine PC Drills	2000/6KB/5	–	20/15.5MB/5	–	1000/32KB/5
Log Telling	3000/43KB/2	120/1MB/2	30/15.5MB/2	80/40KB/2	3000/32KB/2
Repair Maintenance Management	2000/6KB/1	20/1MB/1	–	70/40KB/1	300/32KB/1
Supply Inventory	2500/6KB/1	25/1MB/1	–	40/40KB/1	500/32KB/1
Preventative Maintenance	4000/6KB/1	40/1MB/1	–	70/40KB/1	500/32KB/1
Message Routing	–	15/300KB/1	–	150/40KB/1	–
Watch Bill Scheduling	1500/6KB/3	–	–	40/40KB/3	30/32KB/3
Recruit Planning	1500/6KB/4	24/300KB/4	–	40/40KB/4	–
ROD Deployment	1500/6KB/3	30/300KB/3	10/15.5MB/3	20/40KB/3	–
Qualification System	1000/6KB/1	24/1MB/1	–	24/40KB/1	1000/32KB/1
Training and RPA Tracking	–	10/300KB/1	–	40/40KB/1	–
General Record Storage and Retrieval	1500/6KB/4	48/1MB/4	30/15.5MB/4	–	80/32KB/4
Online Training	1500/43KB/1	12/1MB/1	3/15.5MB/1	–	–
Online Ship Inspection and Exam History	500/6KB/4	6/1MB/4	–	–	60/32KB/4
Crew Leisure Activities	15000/43KB/6	750/1MB/6	200/15.5MB/6	1500/40KB/6	100/32KB/6

Figure 41 - Workload Matrix

2. Timeframes

Timeframes represent a way to group task that occur in similar timeframes during the day. This provides for an appropriate statistical input to OPNET Modeler.

#	Timeframe Description	Timeframe Occurrence
1	Paperwork/Post-WatchBased (PPB)	(0700-0800, 1300-1400, 1900-2100)
2	Watch Based (WB)	(Every Hour for 24hrs)
3	Pre-Watch Based (PB)	(0400-0500, 1000-1100,1600-1700,2200-2300)
4	Pre-Inspection Based (PIB)	(Minus 5 Days prior to inspection)
5	Random Day Based (RDB)	(0700-1630)
6	Random Based (RB)	(24Hr Day with Normalization (Note 1))

Figure 42 - Timeframes Defined

Note 1: The task Crew Leisure Activities occur in the Random Based Timeframe. The Random Based timeframe has a normalization factor to account for activities that would naturally become lower during specific times of the day. These activities include web browsing and using the network for entertainment purposes. Normally during the Mid-Watch, a significant portion of the crew is sleeping; thus a 60% reduction normalization was applied to the calculated load to reduce the loading to reflect real life. Also during the day the crew is busy doing other task areas therefore a 30% reduction normalization was applied to crew activities during this time frame. Last, during meal times a 80% reduction normalization was applied due to the fact that a significant portion of the crew is eating and not doing network activities.

3. Workloads Assumptions

a. Mean Application Frequency for Given Timeframe

The mean frequency for each application is expected to occur for a given task area (e.g. Random Day Based occurs from 0700-1630). These numbers were based on what is needed to support the given task area.

b. Mean Size of Application Loading

OPNET Modeler helped provide this data. The determined load size of each application are listed and described below:

- HTTP: This application supports two sizes 43KB and 6KB. 43KB is considered image browsing and consists of one ASCII page with a constant size of 1KB and 7 large image pictures with a mean size of 6 KB. This equals 43KB per page. 6KB is considered regular browsing and consists of one ASCII page of constant size 1 KB and 5 medium picture files of 1 KB each for a total of 6KB. Each request and response is preceded by a 150-byte header, but will be assumed zero in calculations. OPNET maintains four TCP connections per HTTP client to limit the number of active connections at any given time. HTTP 1.1 persistent protocols is used during the simulation. One SWIPNet DC report sent corresponds to one 6KB ASCII HTTP request.
- FTP: This application supports two mean sizes of 1MB and 300KB for get/puts. The acknowledgement of a put transaction is a constant 512-byte message. In a get transaction, the initial request is a constant 512-byte message. Effective transfer sizes are assumed to be 1MB and 300KB for a get or put.
- Streaming Audio/Video: This assumes high quality video of 128x120 resolution at 10-frames/ sec. and assumes equal and constant bandwidth in both directions (client to server and back). An estimation of a mean streaming video session is 15.5 MB each used to calculate the workloads.
- Email: Supports one size of 40 KB. This assumes the mean size of each person's mail is 20 KB and that you have one sender and one receiver. So you have one upload and one download for a total of two for each email. E-mail is acknowledged by a 16-byte response message, so the effective transfer is 40KB per e-mail sent.

- Database (DB): Supports one size of 32KB per query/entry of the database. Each database query transaction request is a constant 512-byte acknowledgement. Effective transfer per query/entry is assumed a constant 32 KB for this application.

c. *Timeframe#*

All task areas will be mapped to a specific timeframe. The timeframes depicted in Figure 42 are the respective times expected to occur.

d. *Task Areas*

15 task areas were evaluated. It is assumed that all of these tasks will be performed only via handheld wireless units. It is understood the workloads developed from these tasks could occur on wired workstations, but the analysis looks at worst load conditions. Workstations would have the advantage of tying into the 100 Mbps BUS LAN, but the 11 Mbps wireless section is the main focus in this study.

e. *Servers*

Assume that all the tasks will be serviced from a central location. Specifically, the PDC and BDC, which are located in NAV. These services are provided for an assumed 24-hour normal workday. The BDC is disabled to allow analysis of a full workload through one server, the PDC.

f. *Application Services Selection*

It is assumed that only HTTP, FTP, e-mail, Database, and Streaming Audio and Video are needed for all 15 task areas defined earlier. Telnet services loading were assumed to be negligible due to its light loading requirements.

4. *Workloads Plotted*

Results were plotted in Figure 43. This figure shows that the maximum loading occurs between 1900 and 2100. These times will be the focus within the OPNET model.

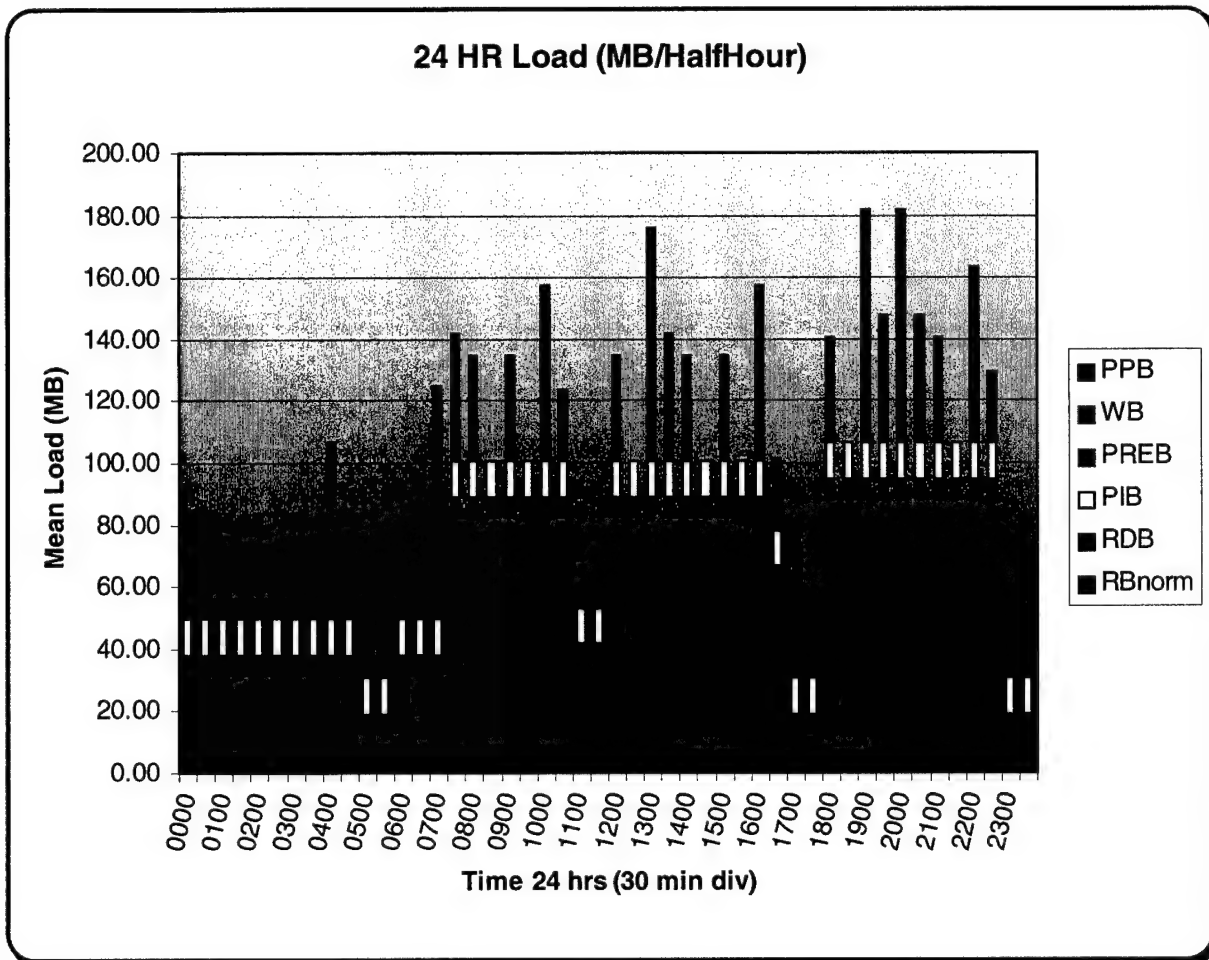


Figure 43 - Twenty-Four Hour Loading Results

5. Workload Grouping

All workloads (i.e., HTTP43KB, HTTP6KB, FTP300KB, FTP1MB, EMAIL 40KB, DB32KB and Streaming Video) are applied simultaneously with a Poisson distribution inter-arrival time (illustrated in Table 5) and are calculated from their mean frequency per half hour.

Application Supported		HTTP	~>	FTP	~>	Streaming	Email	Database
Size (KB)	->	3	6	1000	300	15500	40	32
Mean Total Frequency (per half hour)	->	0	304	6	1	1	15	79
Mean Inter Arrival Time (seconds between events)	->	0	6	293	1,464	1,239	117	23

Table 5 - Inter-Arrival Times

6. Application Grouping

Similar applications are group and applied them to five distinct wireless clients. For example, HTTP6KB and HTTP43KB were applied to the HTTP wireless client.

7. OPNET Simulation Results

This section presents and discusses the results from the OPNET Model with a "x1" (times one) workload applied. This workload is the two hour time period (1900 to 2100) developed in the 24-hour workload graph discussed earlier. Probes were placed throughout the model to collect Link Throughputs, Application Response Times and Throughputs, and Wireless Delay, Load and Throughput and Ethernet Delay. Simulations reach steady state about the 10 minute mark, so this simulation was run to 30 minutes and captures the behavior for a two hour interval described in the workload section. The Full workload was distributed as follows. DB Client was in AP12 (Blue Subnet), EMAIL Client within AP3 (Orange Subnet), VIDEO Client within AP14 (Red Subnet), HTTP Client within AP4 (Gold Subnet), and FTP client within AP13 (Green Subnet).

a. Subnet-to-Subnet Throughput (bps)

Figure 44 depicts the traffic flow between subnets. As shown, the maximum steady state throughput of 1,540,000 bps occurs between the Red to Green Subnet and the Green to Gold Subnet Link. This steady value is reached at the 30-minute

real time point of the simulation. This highest throughput occurs in these link due to placement of the HTTP client within the Gold Subnet and the VIDEO client within the Red Subnet.

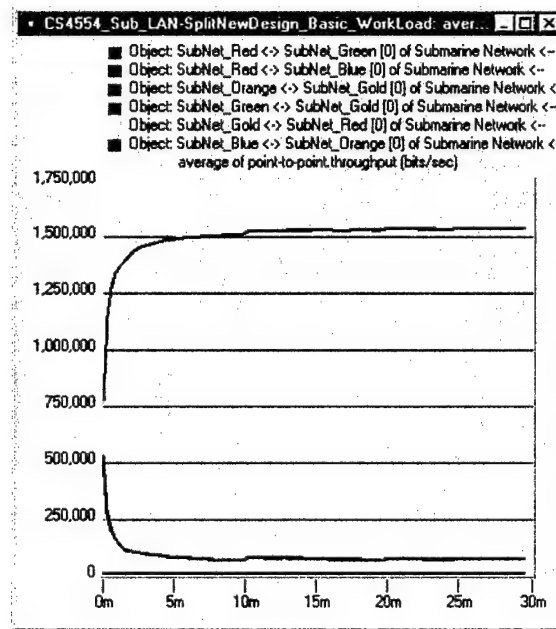


Figure 44 - Subnet-to-Subnet Throughput [bps vs. minutes (m)]

b. Application Mean Response Time

Figure 45 b) illustrates the events (dots) occurring during the simulation. The max response time is three seconds for an FTP download. This makes sense because FTP is the largest file (1MB) within the applications. The Mean (Average) Response Time is also included.

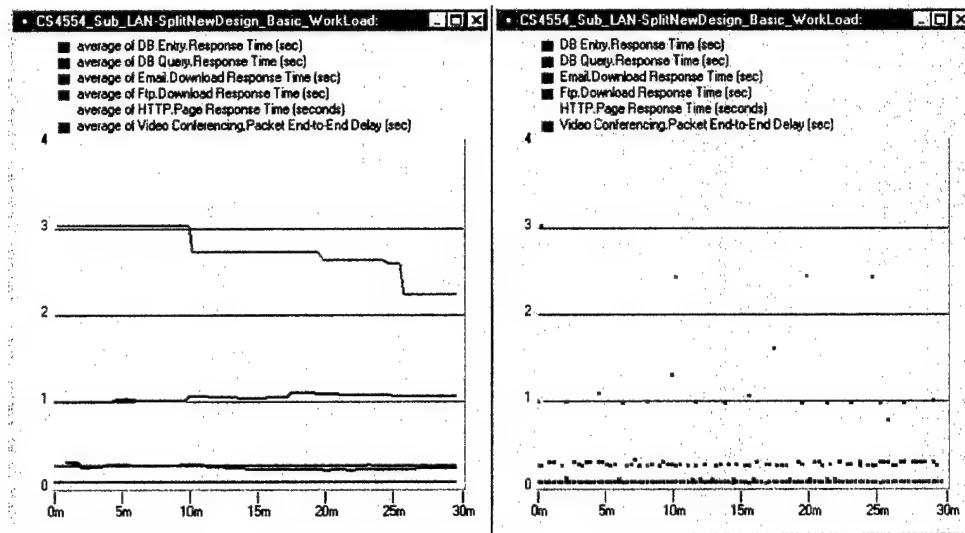


Figure 45 - a) Application Response Time b) Application Response Time

c. Application Load (Wireless Clients)

It can be seen, from Figure 46, that VIDEO and HTTP clients place the largest load on the system. This occurs because HTTP has a very high frequency (or small Inter arrival time) and video because it is sending and receiving 10 frames/sec at 128 by 120 resolution.

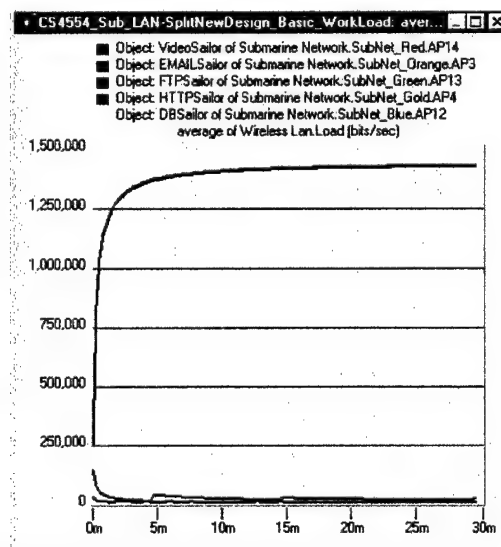


Figure 46 - Application Load From Wireless Clients [bps vs. minutes (m)]

d. Ethernet Delay

The Ethernet Delay is constant at .000376 sec, as shown in Figure 47. This value is similar to the calculated values presented latter in the Analytical Modeling section.

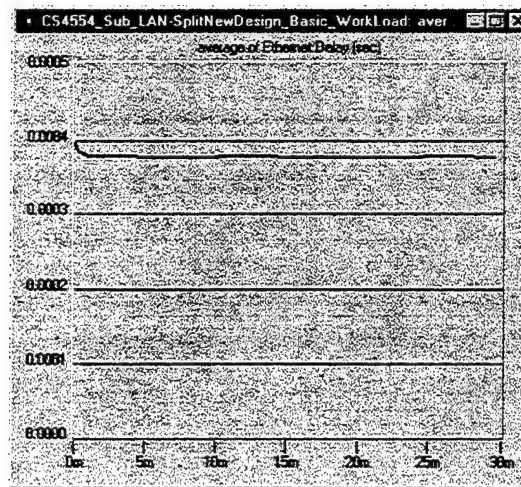


Figure 47 - Ethernet Delay for a Basic Workload

J. ANALYTICAL MODELING

To validate the results generated from the OPNET simulation model of the submarine LAN, an analytical model was developed in order to provide values that could be compared to the values generated by the simulation. Since the submarine LAN utilizes an Ethernet protocol, the mean delay equation for CSMA/CD networks presented in Sadiku and Ilyas' *Simulation of Local Area Networks* [Sadiku95] provided an easy and efficient way to measure the effects of various sizes of data transferred on the network.

1. Definition of Problem

Due to the complexity of the submarine LAN architecture, taking measurements of the entire network proved to become overly tedious. Therefore, in order to simplify the analytical model, the worst case scenarios was chosen to provide a comparison of the two models. The Longest Path scenario relates to the longest path that data must travel

and the maximum nodes encountered. For the submarine LAN, the longest path (see Figure 48) was determined by simulating the data that must travel from the primary server and reach a user at access point 13 within the Gold subnet. Switches A and B were made unavailable within both models (OPNET and Analytical) to represent those components being considered not operational. This forces the data to travel from the primary server through Switch C, out through switch 5, switch 4, switch 3, switch 2, access point 13, and eventually reaching the client. The overall distance is estimated to be 495 feet.

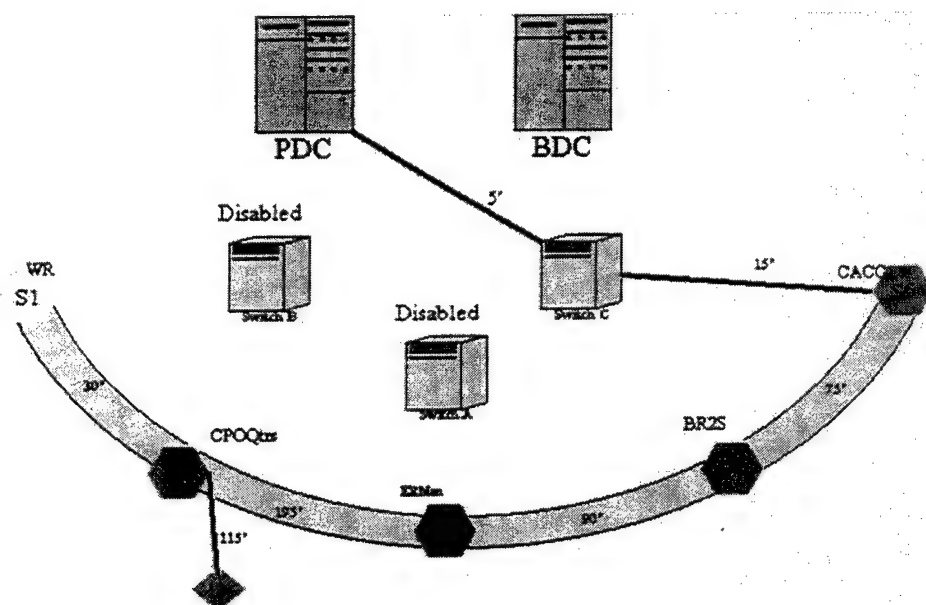


Figure 48 - Longest Path scenario

2. Assumptions of the Analytical Model

The following lists the assumptions made in order to conduct the analytical model:

- ☐ Arrivals at all nodes follow a Poisson arrival rate.
- ☐ Queues are modeled as a M/G/1 [Sadiku95].
- ☐ All stations generate traffic at the same rate.

- Packet Length is equal to the Maximum Segment Size of 64Kbits.
- The transmission medium is assumed to be error free.
- The propagation delay is 5 μ s/km through the transmission medium.
- The wireless components are treated as CSMA/CD connected devices to simplify the calculation vice CSMA/CA
- Longest Path: Switches A and B are non-operational, thus the longest path from the access point within the Green subnet to the server is 495 ft.

3. Approach

The general formulas used to determine the Ethernet Delay E (D) are illustrated in Figure 49. E (D) was calculated for several values of mean arrival rate (λ) (i.e., 20, 40, 60, 80, and 100 packet/sec) applied to one client within AP13. Note: The actual values and computation were performed using a Microsoft Excel spreadsheet. Example calculation: $\lambda = 20$ packets/sec (with Packet Length = 64000 bits/packet):

The mean service time: $E(S)$
 $E(S) = L_p / R = 64000 / 100000000$
 $E(S) = 0.00064$ s
 The mean arrival rate: λ
 $\lambda = 20 \times 64000 = 1,280,000$ bps
 The total arrival rate: $N\lambda$
 $N\lambda = 20 \times 11680 \times 1 = 1,280,000$ bps (longest path)
 The traffic intensity: ρ
 $\rho = \lambda \times E(S) = 1,280,000 \times 0.0006400 = 819.2$
 The end-to-end propagation delay: τ
 $\tau = 1 \times P = 495 / 3.28 \times 10^8 = 7.54 \times 10^{-7}$ s
 For constant packet length: $F(\lambda)$, $E(S^2)$
 $F(\lambda) = 1$, and $E(S^2) = E^2(S) = 4.096 \times 10^{-7}$

Mean Ethernet Delay E(D):

$$E(D) = \frac{\lambda [E(S^2) + (4s + 2)\tau E(S) + 5\tau^2 + 4s(2s - 1)\tau^2]}{2(1 - \lambda [E(S) + \tau + 2s\tau])} - \frac{(1 - s^{-2} \lambda \tau)(s + \lambda \tau - 3 \lambda \tau s)}{\lambda s [F(\lambda) s^{-(1 + \lambda \tau)} + s^{-2} \lambda \tau - 1]} + 2s + E(S) + \frac{\tau}{3} = .00032 \text{ sec}$$

Figure 49 - Calculation of Mean Ethernet Delay E(D)

Using the mean delay equation, delay values were calculated for the Longest Path scenario. The only input parameter that was varied during the computation of values was the total arrival rate of the data.

In order to provide a comparison set of values that could be compared with the OPNET and the analytical results, a C simulation program for CSMA/CD LAN's (Appendix B of the Sadiku and Ilyas text [Sadiku95]) was compiled and ran. The parameters (i.e. Distance, #nodes, bus speed, etc) for the Longest Path scenarios were input to the program and the results for the Mean Ethernet Delay, $E(D)$, were calculated.

4. Analytical Results

The results of the analytical model were computed using a Microsoft Excel spreadsheet. These values are compared in Table 6 to the OPNET results and C simulation program.

(packets/sec)	$E(D)$ in sec [Eqtn Shown]	$E(D)$ in sec [Sadiku's Program]	$E(D)$ in sec OPNET
0	0	0	0
20	0.000320324	0.000335294	0.00040523
40	0.000320129	0.000342993	0.000404641
60	0.000320064	0.000353308	0.000405748
80	0.000320032	0.000364436	0.000406155
100	0.000320012	0.00037772	0.00040702

Table 6 - Longest Path $E(D)$ Comparison Table

Next these values were plotted in order to graphically represent the results (Figure 50). As the graphs indicate, there is a great deal of similarity between the values of the analytical model and the Sadiku and Ilyas CSMA/CD simulation program [Sadiku95] values. Additionally, both of these results nearly correspond to the results from the OPNET simulator. The average $E(D)$ between these three methods for the Longest Path scenario is approximately .00036825 sec.

Longest Path Delay vs Arrival Rate

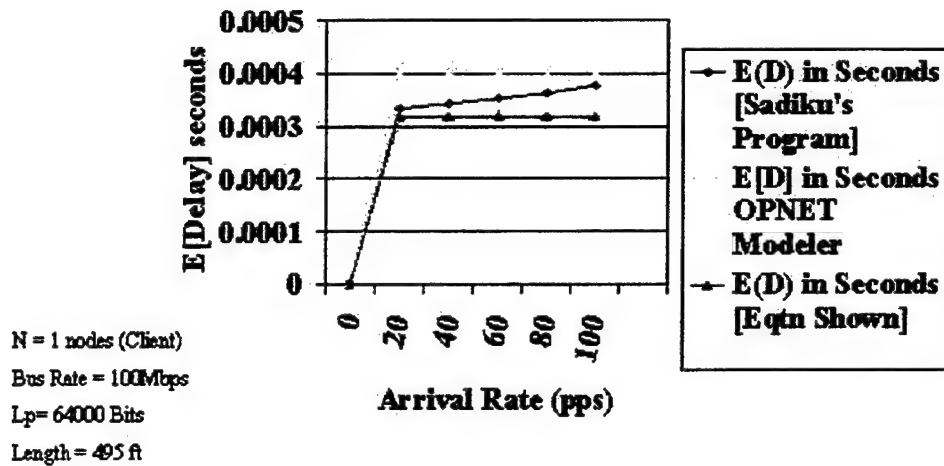


Figure 50 - Delay vs. Arrival Rate

The differences between calculated and the actual OPNET model is attributed to the simplifying assumptions made for CSMA/CA type networks.

K. SENSITIVITY ANALYSIS

1. Workloads Applied

The following scenarios were run to stress the system, by increasing the number of clients/workload, until the system failed. It is shown that VIDEO streaming clients severely limits the overall number of clients the system can handle. So to increase the number of HTTP, FTP, DB, and EMAIL clients, VIDEO was excluded or varied to see were the system performance degraded significantly. FTP Response is shown in Figure 51. The following workloads were applied:

- "Basic_Workload" – Is the workload developed within the Workload Section of this chapter. Includes one client of VIDEO, HTTP, FTP, EMAIL, and DB (5 Wireless Clients total).

- “x5_VF” Workload – Increases the Basic Workload by five (25 Clients total)
- “x10_x8Vid” – Increases HTTP, FTP, EMAIL, and DB by 10 and Video by eight. (48 Clients total). At this point the degradation of the MAC/TCP layers were seen. Buffer overflows started causing dropped packets and the simulation would only run for 15 minutes (model time) and took 14 hours to finish the simulation.
- “x10_VF” – Increases the Basic workload by a factor of 10. (50 Clients total). This simulation resulted in a simulation failure due to 10 video clients overloading the system and causing excessive buffer overflows.
- “x10_NoVid” – At this point the simulations was changed to include no video. This simulation is a factor of ten of the basic workload without VIDEO. (40 Clients Total).
- “x19_NoVid” – This idea continues (76 Clients total).
- “x38_NoVid” – Represents 152 wireless clients excluding video.
- “x57_NoVid” – Represents 228 wireless clients excluding video. The maximum delay for each packet in this workload approaches 35 seconds at the 30 minute real time point of the simulation. This delay is unacceptable and is a result of having too many clients that are overloading the system. This delay is similar to the degradation seen in the “x10_x8Vid” workload that provided a huge VIDEO load on the system.

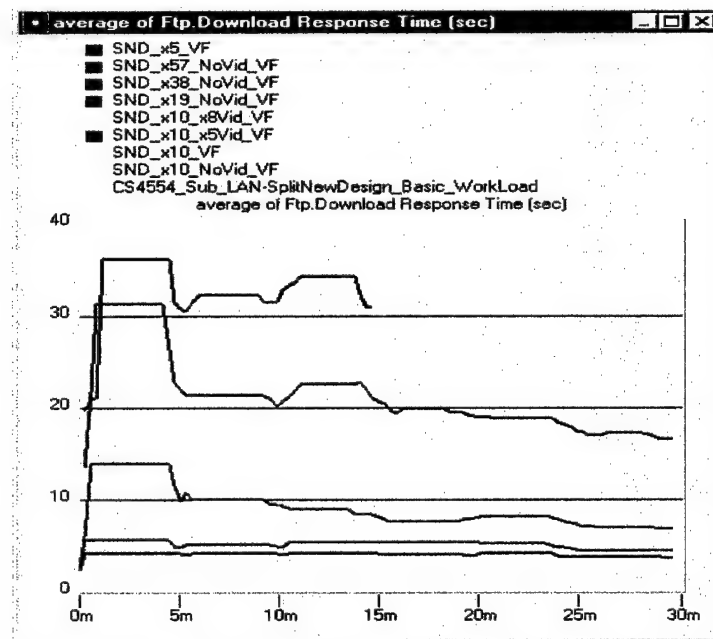


Figure 51 - FTP Response Time (sec vs. minutes) [All Simulation Runs]

The Ethernet delay (Figure 52) for the full system centers around .0004 seconds and is fairly constant regardless of the workload applied. This is attributed to the longest path effect and constant distance (i.e. fixed wireless Clients assumption) that anchors the system delay. This value is the expected delay for any simulation run.

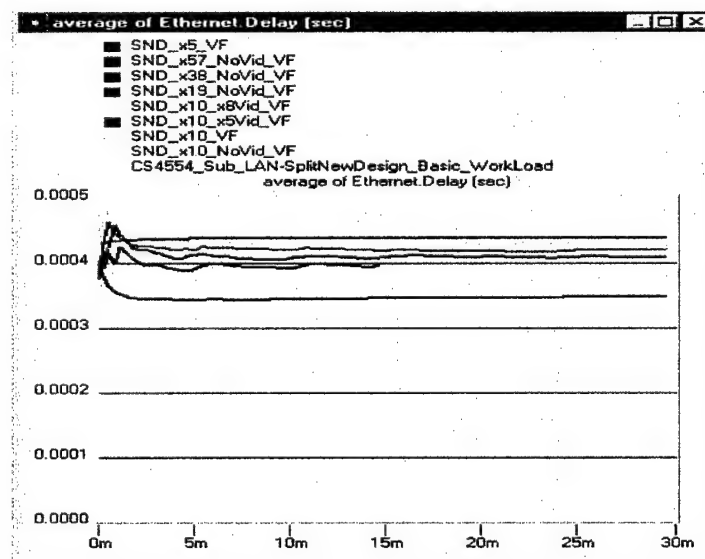


Figure 52 - Ethernet Delay E(D) (sec vs. minutes) [All Simulation Runs]

As shown in Figure 53 the simulation with five video clients starts to become unstable as End-to-End delay approaches infinity. Whereas the “x38_NoVid” remains stable, but with borderline unacceptable response times.

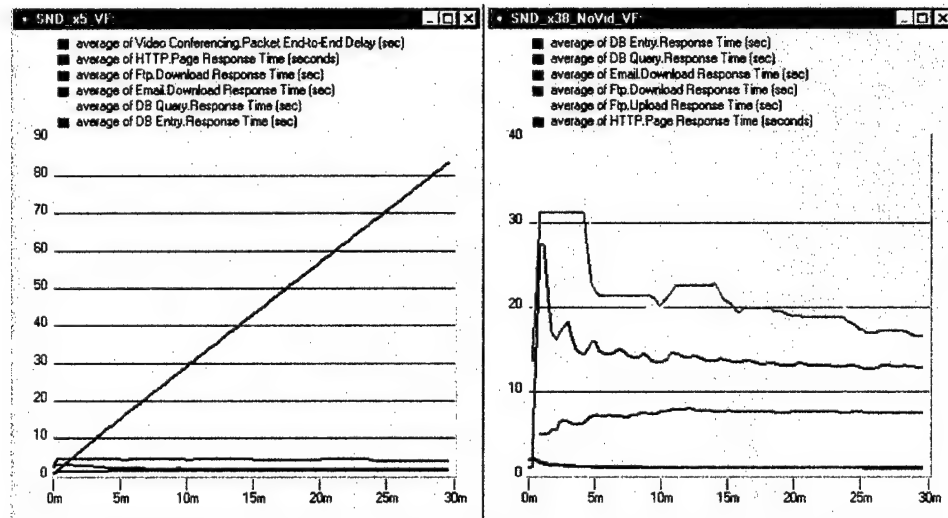


Figure 53 - Response Times a) x5 Workload b) x38 with No Video

L. CHAPTER SUMMARY

This simulation of the proposed NTDPS LAN to determine the feasibility of deploying a virtual DC system based on this LAN’s loading capacity and the 14 other tasks that could be assumed would be running at various times throughout the day. Based on this study this Network could easily handle this application. It has been shown during this study that up to 152 wireless clients with each running the full fifteen tasks workload, as shown in Figure 41, can run without acceptable degradation. If this virtual DC application integrates streaming video on this system then no more than five could be transmitting this video stream at one time. These five could be improved to a higher value if these bandwidth intensive streams could be moved to a hardwired connection and thus take advantage of the 100Mbps line. This study has also shown that this system has an expected worst case delay, of .00036825 seconds, is fairly constant regardless of the proposed workloads applied. Also based on the normal days workload, as shown in Figure 43, the peak workload would occur between 1900-2100 giving more room on the

network for virtual DC drills that normally occur during the daytime. Ultimately this study has shown that this system could provide support for a real casualty that could of course occur anytime.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. RECOMMENDATIONS AND CONCLUSION

A. SUMMARY

This thesis explored current software and hardware technologies to determine their effectiveness on today's submarines. It looked at a five year culmination of research conducted by the NPS wireless research group and summarized this research into six key areas: wireless theory, standards, hardware, software development, testing and system requirements. This thesis then advanced these concepts in the area of software development and network simulation testing.

Software development includes analyzing the day-to-day routine onboard a submarine to determine what task could be improved with a wireless based application design. Fifteen tasks were identified that could benefit directly from this type of application. One important task, Casualty and Damage control, was selected as the model for implementation. To help develop a design the DC communication model from the USS Batfish was analyzed. From this model a set of design requirements were developed that specified the minimum requirements the system would need. From these requirements SWIPNet was developed. SWIPNet is a three console, Java based, multithreaded, multicast socket driven, application. It is designed to provide standard DC and crew reports to anywhere on a ship that has wireless coverage. This application uses multicast sockets to create a virtual communication channel similar to the sound powered phone communications used on submarines currently. Future application features include the ability to provide streaming video from cameras attached to NFTI and an ability to reference technical drawings virtually to provide more effective casualty control. A database connection model was partially implemented to provide persistent storage and reliability in case of a system wide power outage when battery backups fail. For the persistent storage model a Microsoft Access 2000 database to store system critical information was developed and JDBC (Java Database Connectivity) was utilized to connect to SWIPNet for initialization.

After the completion and demonstration of SWIPNet it was important to know if this application would function effectively in a real wireless submarine environment. To accomplish this in the most precise manner OPNET Modeler 7.0B was used to produce a simulation of the NTDPS, a proposed Virginia Wireless network, in the presence of a SWIPNet type load. This simulation had several parts. The first part involved a slight modification to the original design obtained from NAVSEA Code450 to adhere to the OPNET Modeler criteria. The next step included designing a submarine cross section and laying out the network components (Switches, Servers, Subnets etc). The next step involved the creation of a detailed 24 hour workload that includes a SWIPNet type load and uses various combinations of HTTP, FTP, EMAIL, DB and Audio and Video to accomplish their tasks. This full workload was then applied to the built OPNET simulation. Subnet-to-Subnet throughput and application response time graphs show that SWIPNet would work well on the NTDPS. Sensitivity analysis was also performed to determine when this system would fail by progressively increasing the full workload factor. This analysis showed that the system performs satisfactorily up to 152 wireless clients (each carrying a full workload). To provide an extra layer of validity to the simulation an analytical model was developed for the longest path scenario and the Ethernet delay was compared. The results showed the simulation to be accurate based on similar Ethernet delays.

B. RECOMMENDATIONS FOR FUTURE WORK

SWIPNet is designed to become a deployable virtual DC application. Currently the application is deployable yet it lacks a "full" implementation of three features that this thesis had originally set out to complete. The first includes persistent storage to provide a robust solution against a network power outage and a boat specific initialization sequence. Currently a database and the connection classes (JDBCBridge and Initialize) have been created, but not linked within the SWIPNet code. A recommendation for future work is to use these designed components and finish the linkage to the SWIPNet application. The second feature would be to finish the

integration of a NFTI camera video stream into the sender and listener consoles and the last feature is the displaying of technical drawings to help provide isolation of components in a casualty.

Other areas that could be explored, is to provide a model such as XML for the persistent storage model and a Server Side connection model (e.g. Servlets, JSP's and Enterprise JavaBeans) described in Chapter V. These models could potentially replace the relational database (Microsoft Access 2000) and JDBC models currently designed. Also the OPNET Modeler simulation performed in Chapter VI, could be updated as the more design decisions have been made concerning the NTDPS on the Virginia class wireless network.

C. FINAL CONCLUSION

Access to information is the key for efficient communications onboard a submarine. We must continue to reevaluate our systems to determine the best solution. Technology has produced COTS based wireless components that give us the ability to access information in a mobile environment. It provides us more flexibility and increases information flow at a low cost. This thesis takes advantage of this technology by developing Java based application, known as SWIPNet. SWIPNet can be used to provide more efficient DC communications and quicker response times. This improved DC model can ultimately make our submarines safer.

SWIPNet is an example of what we can do with a wireless system. It illustrates forward thinking of current task and procedures. This application and others like it are the future of mobile computing. It is imperative to place these tools into the hands of our personnel today to create a powerful dynamic work environment to meet our demands of tomorrow.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – SELECTED SOURCE CODE

Package: swipNet.control

Class: DCNet

```
package swipNet.control;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.border.*;
import swipNet.utility.Initialize;

/**
 * DCNet class provides and configuration and launching display,
 * it has the ability to configure the station location,
 * multicast address and launch a Client (sender) and/or
 * Server(listener)<p>
 * @author      LT William G. Wilkins
 * @version 1.0
 */

public class DCNet extends JApplet {

    boolean isStandalone = false;

    /**controls first time initialization from user preferences
    */
    protected boolean firstTimeC = true;

    /**controls first time initialization from user preferences
    */
    protected boolean firstTimeS = true;

    /**Object to initialize the gui comboboxes
    */
    protected Initialize iz = new Initialize();

    /**Client Applet that is launched from DCNet
    */
    protected Client c;

    /**Server Applet that is launched from DCNet
    */
    protected Server s;

    /**Position Dividers
    */
    Divider divOne =new Divider();
    Thread threadOne = new Thread(divOne);

    JToggleButton toggleButtonClient = new JToggleButton();
    JToggleButton toggleButtonServer = new JToggleButton();
    JPanel cPanel = new JPanel();  JPanel sPanel = new JPanel();
    JFrame cNewFrame = new JFrame();  JFrame sNewFrame = new JFrame();
    JLabel jLabelOne = new JLabel();
    JComboBox jComboBoxOne = new JComboBox(iz.dccCasLocArray);
    JLabel jLabelTwo = new JLabel();
    JComboBox jComboBoxTwo = new JComboBox(iz.multicastChoices);
    JLabel jLabelThree = new JLabel();
    JComboBox jComboBoxThree = new JComboBox(iz.multicastChoices);
```

```

JLabel jLabelFour = new JLabel();
JComboBox jComboBoxFour = new JComboBox(iz.dccCasLocArray);
JLabel jLabelFive = new JLabel();
JComboBox jComboBoxFive = new JComboBox(iz.multicastChoices);
JLabel jLabelSix = new JLabel();
JComboBox jComboBoxSix = new JComboBox(iz.multicastChoices);
JLabel jLabelSeven = new JLabel();
JComboBox jComboBoxSeven = new JComboBox(iz.multicastChoices);

/** Constructor of DCNet*/
public DCNet() {
    this.setName("");
}
/** Constructor of DCNet
 * @param args receive name from main()
 */
public DCNet(String [] args){
    this.setName("");
}

/** Initialize DCNet*/
public void init() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**
 * jbinit(), used in Jbuilders gui builder interface, creates the GUI
 */
private void jbInit() throws Exception {
    this.getContentPane().setSize(1010,100);
    this.getContentPane().setLayout(new GridLayout(1,0));

    //set preferences
    cNewFrame.setSize(380,720);
    sNewFrame.setSize(1060,960);
    toggleButtonClient.setText("<html><font size=3
color=white><bold>LAUNCH</font></html>");
    toggleButtonServer.setText("<html><font size=3
color=white><bold>LAUNCH</font></html>");
    cPanel.setBackground(new java.awt.Color(166,162,187));
    cPanel.setBorder(BorderFactory.createEtchedBorder());
    cPanel.setToolTipText("Create Casualty Channels to Send on");
    cPanel.setLayout(null);
    sPanel.setBackground(new java.awt.Color(177, 156, 176));
    sPanel.setBorder(BorderFactory.createEtchedBorder());
    sPanel.setToolTipText("Create Casualty Channels to Listen on");
    sPanel.setLayout(null);
    jLabelOne.setFont(new java.awt.Font("Serif", 1, 12));
    jLabelOne.setForeground(Color.white);
    jLabelOne.setText("SET Location");
    jLabelTwo.setFont(new java.awt.Font("Serif", 1, 12));
    jLabelTwo.setForeground(Color.white);
    jLabelTwo.setText("DC Channel");
    jLabelThree.setFont(new java.awt.Font("Serif", 1, 12));
    jLabelThree.setForeground(Color.white);
    jLabelThree.setText("SHIP Channel");
    jLabelFour.setFont(new java.awt.Font("Serif", 1, 12));
    jLabelFour.setForeground(Color.white);
    jLabelFour.setText("SET Location");

```

```

jLabelFive.setFont(new java.awt.Font("Serif", 1, 12));
jLabelFive.setForeground(Color.white);
jLabelFive.setText("DC Channel 1");
jLabelSix.setFont(new java.awt.Font("Serif", 1, 12));
jLabelSix.setForeground(Color.white);
jLabelSix.setText("DC Channel 2");
jLabelSeven.setFont(new java.awt.Font("Serif", 1, 12));
jLabelSeven.setForeground(Color.white);
jLabelSeven.setText("SHIP Channel");
Insets insetsC = cPanel.getInsets();
jLabelOne.setBounds(5 + insetsC.left, 0 + insetsC.top, 75, 35);
jComboBoxOne.setBounds(85 + insetsC.left, 7 + insetsC.top, 95, 20);
jComboBoxOne.setSelectedIndex(0);
jLabelTwo.setBounds(5 + insetsC.left, 34 + insetsC.top, 75, 35);
jComboBoxTwo.setBounds(85 + insetsC.left, 41 + insetsC.top, 95, 20);
jComboBoxTwo.setSelectedIndex(0);
jLabelThree.setBounds(225 + insetsC.left, 34 + insetsC.top, 80, 35);
jComboBoxThree.setBounds(310 + insetsC.left, 41 + insetsC.top, 95, 20);
jComboBoxThree.setSelectedIndex(2);
toggleButtonClient.setBounds(190 + insetsC.left, 5 + insetsC.top, 300, 25);
toggleButtonClient.setBorder(BorderFactory.createEtchedBorder());
toggleButtonClient.setBackground(new Color(143,138,170));
Insets insetsS = sPanel.getInsets();
jLabelFour.setBounds(5 + insetsS.left, 0 + insetsS.top, 75, 35);
jComboBoxFour.setBounds(85 + insetsS.left, 7 + insetsS.top, 95, 20);
jComboBoxFour.setSelectedIndex(0);
jLabelFive.setBounds(5 + insetsS.left, 34 + insetsS.top, 80, 35);
jComboBoxFive.setBounds(85 + insetsS.left, 41 + insetsS.top, 80, 20);
jComboBoxFive.setSelectedIndex(0);
jLabelSix.setBounds(170 + insetsS.left, 34 + insetsS.top, 80, 35);
jComboBoxSix.setBounds(250 + insetsS.left, 41 + insetsS.top, 80, 20);
jComboBoxSix.setSelectedIndex(1);
jLabelSeven.setBounds(335 + insetsS.left, 34 + insetsS.top, 80, 35);
jComboBoxSeven.setBounds(415 + insetsS.left, 41 + insetsS.top, 80, 20);
jComboBoxSeven.setSelectedIndex(2);
toggleButtonServer.setBounds(190 + insetsS.left, 5 + insetsS.top, 300, 25);
toggleButtonServer.setBorder(BorderFactory.createEtchedBorder());
toggleButtonServer.setBackground(new Color(159,134,158));

//add
this.getContentPane().add(cPanel);
cPanel.add(jLabelOne);
cPanel.add(jComboBoxOne);
cPanel.add(jLabelTwo);
cPanel.add(jComboBoxTwo);
cPanel.add(jLabelThree);
cPanel.add(jComboBoxThree);
cPanel.add(toggleButtonClient);
//add
this.getContentPane().add(sPanel);
sPanel.add(jLabelFour);
sPanel.add(jComboBoxFour);
sPanel.add(jLabelFive);
sPanel.add(jComboBoxFive);
sPanel.add(jLabelSix);
sPanel.add(jComboBoxSix);
sPanel.add(jLabelSeven);
sPanel.add(jComboBoxSeven);
sPanel.add(toggleButtonServer);

} //end jbInit

/** start() the applet, adds listeners to buttons to launch a sender or
listener based on user
location and multicast address input */

```

```

public void start() {
    toggleButtonClient .addItemListener(
        new ItemListener() {
            public void itemStateChanged(ItemEvent e)
            {
                try { //Put Desire Action Here

                    if (e.getItemSelectable()==toggleButtonClient ){

                        if (e.getStateChange() ==
ItemEvent.SELECTED){
                            if(firstTimeC == true){
                                String holdString[]
                                ={(String)jComboBoxOne.getSelectedItem(),
                                (String)jComboBoxTwo.getSelectedItem(),
                                (String)jComboBoxThree.getSelectedItem()}} ;

                                c =new Client(holdString);
                                cNewFrame.getContentPane().add(c,
BorderLayout.CENTER); //Client
                                cNewFrame.setTitle((String)jComboBoxOne.getSelectedItem()
                                + ", Casualty Traffic on "
                                +
                                (String)jComboBoxTwo.getSelectedItem()
                                +" Ship Traffic on " +
                                (String)jComboBoxThree.getSelectedItem());
                                cNewFrame.setLocation(0,100);
                                c.init();
                                c.start();
                                cNewFrame.addWindowListener(
                                    new WindowAdapter(){
                                        public void windowClosing(
WindowEvent e )
                                        {

                                            cNewFrame.setVisible(false);

                                            toggleButtonClient.setText(
                                                "<html><font size=3
color=white><bold>reLAUNCH</font></html>");

                                            toggleButtonClient.setSelected(false);

                                            }
                                        }
                                    );
                                }
                                cNewFrame.setVisible(true);
                                toggleButtonClient .setText(
                                    "<html><font size=3
color=white><bold>HIDE</font></html>");
                                firstTimeC = false;
                                } //end inner if
                                else{
                                    cNewFrame.setVisible(false);
                                    toggleButtonClient .setText(
                                        "<html><font size=3
color=white><bold>reLAUNCH</font></html>");
                                    } //end else
                                } //end outer if
                                } //end try
                                catch (Exception ex){

```

```

        ex.printStackTrace();
    }
    }); //end inner class and method call

toggleButtonServer.addItemListener(
    new ItemListener() {
        public void itemStateChanged(ItemEvent e)
        {
            try { //Put Desire Action Here

                if (e.getItemSelectable()==toggleButtonServer){
                    if (e.getStateChange() ==
ItemEvent.SELECTED){
                        if(firstTimeS == true){
                            String holdString[]
= {(String)jComboBoxFour.getSelectedItemAt(),
(String)jComboBoxFive.getSelectedItemAt(),
(String)jComboBoxSix.getSelectedItemAt(),
(String)jComboBoxSeven.getSelectedItemAt()) ;

                            s =new Server(holdString);

                            sNewFrame.getContentPane().add(s,
BorderLayout.CENTER); //Server
sNewFrame.setTitle((String)jComboBoxFour.getSelectedItemAt()
+ ", Casualty Traffic on "
+
(String)jComboBoxFive.getSelectedItemAt()
+ " and " +
(String)jComboBoxSix.getSelectedItemAt()
+ " Ship Traffic on " +
(String)jComboBoxSeven.getSelectedItemAt());
sNewFrame.setLocation(0,100);
sNewFrame.setVisible(true);
sNewFrame.addWindowListener(
    new WindowAdapter(){
        public void windowClosing(
WindowEvent e )
        {

sNewFrame.setVisible(false);

toggleButtonServer.setText(
    "<html><font size=3
color=white><bold>reLAUNCH</font></html>");
toggleButtonServer.setSelected(false);

        }
    }
);
s.init();
s.start();
threadOne.start();//Start Thread to
Set Dividers
}
sNewFrame.setVisible(true);
toggleButtonServer.setText(
    "<html><font size=3
color=white><bold>HIDE</font></html>");
firstTimeS = false;

```



```

        } //end inner if
        else{
            sNewFrame.setVisible(false);
            toggleButtonServer.setText(
                "<html><font size=3
color=white><bold>reLAUNCH</font></html>");
        } //end else
    } //end outer if
} //end try
catch (Exception ex){
    ex.printStackTrace();
}
} //end inner class and method call

} //end Start

/**Stop the applet - not implemented*/
public void stop() {

}

/**Destroy the applet, cleanup of threads and sockets of client and server
applets*/
public void destroy() {
    c.destroy();
    s.destroy();
}

private class Divider implements Runnable{ //Inner Class to Set Dividers
    public Divider(){}
    public void run(){
        s.setDivider();
    }
}

/**Entry Point, used to initialize the applet as an application, allows
program to function as
* an applet or an application
*
*@param args[] An array of methods from the command line arguments
*@return void
*/
public static void main(String[] args) {

    DCNet applet = new DCNet(args);
    applet.isStandalone = true;
    JFrame frame = new JFrame();
    frame.setTitle("DCNet                                     Send DC
Channel Reports"
    +
    +
    +
    Channels");
    frame.getContentPane().add(applet, BorderLayout.CENTER);

    applet.init();
    applet.start();

    /**Stop process on window close*/
    frame.addWindowListener(
        new WindowAdapter(){
            public void windowClosing( WindowEvent e )

```

```

        {
            System.exit(0);
        }
    }
);

try{
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation(0, 0);
    frame.setSize(1010,100);
    frame.setVisible(true);
}
catch(Exception e){

}
}
// static initializer for setting look & feel
static {
    try {
        UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
    );
    }
    catch (Exception e) {}
}
}
}

```

Class: Client object

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.border.*;
import java.net.*; //Socket objects
import java.util.*; //For Date
import swipNet.utility.*; //Post Office
import swipNet.dcObjects.*; //Fire, etc
import swipNet.shipObjects.*; //ShipStatus, etc
import swipNet.gui.*; //for control bar gui

/**
 * Client subscribes to two multicast address and sends Casualty and Ship
 * reports within one Communicate thread; It creates a client gui by adding
 * gui objects such as Firegui, etc
 * and uses a Postoffice object to send, Fire, Flooding objects, etc.<p>
 * @author LT William G. Wilkins
 * @version 1.0
 */

public class Client extends JApplet {

    boolean isStandalone = false;

    //Creation of GUI Objects
    //Fire
    //Level 1
    JTabbedPane casualtyTabPanel = new JTabbedPane();
    //Level 2
    Fire fireOne = new Fire(false); //Set false since a client
    Flooding floodingOne = new Flooding(false);
    HydRupture hydRuptureOne = new HydRupture(false);
    AirRupture airRuptureOne = new AirRupture(false);
    HotRun hotRunOne = new HotRun(false);
    FastLeak fastLeakOne = new FastLeak(false);
    SlowLeak slowLeakOne = new SlowLeak(false);
    StmRupture stmRuptureOne = new StmRupture(false);
    RxScram rxScramOne = new RxScram(false);
    RadSpill radSpillOne = new RadSpill(false);
    ShipStatus ss = new ShipStatus(false);
    EngineeringStatus es = new EngineeringStatus(false);
    CompartmentRigs cr = new CompartmentRigs(false);
    ShipAtmospheres sa = new ShipAtmospheres(false);

    //Misc Creation
    ControlBarGui bar = new ControlBarGui();
    BorderLayout borderLayout1 = new BorderLayout();
    Communicate commOne;
    Thread clientOne;

    /**
     * Client() constructor, used if no arguments are not supplied from user
     */
    public Client() {
        this.setName("Casualty Traffic on " + "228.7.5.4" + " Ship Traffic on " + "
228.7.5.6");

        fireOne.setOwner("Scene", "228.7.5.4"); //Nothing special about these
multicast address
        fireOne.fireGui.host.setText("Send on 228.7.5.4");
        floodingOne.setOwner("Scene", "228.7.5.4");
        floodingOne.floodingGui.host.setText("Send on 228.7.5.4");
        hydRuptureOne.setOwner("Scene", "228.7.5.4");
    }
}
```

```

hydRuptureOne.hrGui.host.setText("Send on 228.7.5.4");
airRuptureOne.setOwner("Scene", "228.7.5.4");
airRuptureOne.arGui.host.setText("Send on 228.7.5.4");
hotRunOne.setOwner("Scene", "228.7.5.4");
hotRunOne.hrGui.host.setText("Send on 228.7.5.4");
fastLeakOne.setOwner("Scene", "228.7.5.4");
fastLeakOne.flGui.host.setText("Send on 228.7.5.4");
slowLeakOne.setOwner("Scene", "228.7.5.4");
slowLeakOne.slGui.host.setText("Send on 228.7.5.4");
stmRuptureOne.setOwner("Scene", "228.7.5.4");
stmRuptureOne.srGui.host.setText("Send on 228.7.5.4");
rxScramOne.setOwner("Scene", "228.7.5.4");
rxScramOne.rsGui.host.setText("Send on 228.7.5.4");
radSpillOne.setOwner("Scene", "228.7.5.4");
radSpillOne.rsGui.host.setText("Send on 228.7.5.4");

ss.setOwner("Scene", "228.7.5.6");
ss.ssGui.host.setText("Send on 228.7.5.6");
es.setOwner("Scene", "228.7.5.6");
es.esGui.host.setText("Send on 228.7.5.6");
cr.setOwner("Scene", "228.7.5.6");
cr.crGui.host.setText("Send on 228.7.5.6");
sa.setOwner("Scene", "228.7.5.6");
sa.saGui.host.setText("Send on 228.7.5.6");

commOne = new Communicate("228.7.5.4", "228.7.5.6");
clientOne = new Thread(commOne);
}
/**
 * Client() constructor, used if arguments are supplied from user
 *
 * @param args used to set gui name and owner if arguments are supplied from
 * user
 */
public Client(String [] args){

    this.setName(args[0] + ",Casualty Traffic " +args[1] +"Ship Traffic " +
args[2]);

    fireOne.setOwner(args[0],args[1] );
    fireOne.fireGui.host.setText(args[0] + ", Send on " + args[1]);
    floodingOne.setOwner(args[0],args[1] );
    floodingOne.floodingGui.host.setText(args[0] + ", Send on " + args[1]);
    hydRuptureOne.setOwner(args[0],args[1] );
    hydRuptureOne.hrGui.host.setText(args[0] + ", Send on " + args[1]);
    airRuptureOne.setOwner(args[0],args[1] );
    airRuptureOne.arGui.host.setText(args[0] + ", Send on " + args[1]);
    hotRunOne.setOwner(args[0],args[1] );
    hotRunOne.hrGui.host.setText(args[0] + ", Send on " + args[1]);
    fastLeakOne.setOwner(args[0],args[1] );
    fastLeakOne.flGui.host.setText(args[0] + ", Send on " + args[1]);
    slowLeakOne.setOwner(args[0],args[1] );
    slowLeakOne.slGui.host.setText(args[0] + ", Send on " + args[1]);
    stmRuptureOne.setOwner(args[0],args[1] );
    stmRuptureOne.srGui.host.setText(args[0] + ", Send on " + args[1]);
    rxScramOne.setOwner(args[0],args[1] );
    rxScramOne.rsGui.host.setText(args[0] + ", Send on " + args[1]);
    radSpillOne.setOwner(args[0],args[1] );
    radSpillOne.rsGui.host.setText(args[0] + ", Send on " + args[1]);

    ss.setOwner(args[0],args[2] );
    ss.ssGui.host.setText(args[0] + ", Send on " + args[2]);
    es.setOwner(args[0],args[2] );
    es.esGui.host.setText(args[0] + ", Send on " + args[2]);
    cr.setOwner(args[0],args[2] );
    cr.crGui.host.setText(args[0] + ", Send on " + args[2]);

```

```

        sa.setOwner(args[0],args[2] );
        sa.saGui.host.setText(args[0] + ", Send on " + args[2]);

        commOne = new Communicate(args[1], args[2]);
        clientOne = new Thread(commOne);
    }

    /**
     * init() - nest jbinit(), to catch exception and use Jbuilders gui builder
    */
    *interface
    */
    public void init() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * jbinit(), used in Jbuilders gui builder interface, creates the GUI
     *
     * @return void
     */
    private void jbInit() throws Exception {
        this.setGUIPreferences();
        this.addGuiComponents();
        this.setJMenuBar(bar);
    }

    /**
     * start the communicate thread
     *
     * @return void
     */
    public void start() {
        clientOne.start();
    }

    /**
     * Communicate - inner class that sends onjects to a server class
     */
    private class Communicate implements Runnable{

        int portNumber = 5000;
        MulticastSocket socket;
        InetAddress addressOne;
        InetAddress addressTwo;

        PostOffice po = new PostOffice();
        String multicastAddressCasualtyOne= "228.7.5.4";//Default for Casualty
        Objects
        String multicastAddressCasualtyTwo= "228.7.5.6";//Default for ShipObjects

        String threadName;
        int sendNumFire = 1;int sendNumFlooding = 1;int sendNumHrup = 1;int
        sendNumAr = 1;
        int sendNumHrun = 1;int sendNumFl = 1;int sendNumSl = 1;int sendNumSr = 1;
        int sendNumRscram = 1;int sendNumRspill = 1;int sendNumSs = 1;int sendNumEs
        = 1;
        int sendNumCr = 1;int sendNumSa = 1;

    }

    /**
     * Communicate(String mcAddressOne, String mcAddressTwo) - contructor, with

```

```

* String input address
* @param mcAddressOne the address the casualty data is sent on
* @param mcAddressTwo the address the ship data is sent on
*/
public Communicate(String mcAddressOne, String mcAddressTwo){

    multicastAddressCasualtyOne = mcAddressOne;
    multicastAddressCasualtyTwo = mcAddressTwo;

}

/**
*Adds listeners to each Button within each gui, and if pressed packages the
* current state of user input
* to the gui interface and sends the data using a postoffice object.
* @return void
*/
public void run()
{

    try
    {
        socket = new MulticastSocket(portNumber);
        addressOne = InetAddress.getByName(multicastAddressCasualtyOne);
        addressTwo = InetAddress.getByName(multicastAddressCasualtyTwo);
        socket.joinGroup(addressOne);
        socket.joinGroup(addressTwo);

        threadName = "Thread One";
        Thread.currentThread().setName(threadName);

        fireOne.fireGui.jButton.addActionListener(//Listeners to
send info
        new ActionListener(){
            public void actionPerformed( ActionEvent e )
            {
                try {//Put Desire Action Here

fireOne.setData(addressOne.getLocalHost().toString(),returnCurrentTime(),"Messa
ge");

                fireOne.setStatusFromGui();
                po.sendMulticastPacket(fireOne,
socket,addressOne);

fireOne.fireGui.directLinkTextArea.append(sendNumFire++ + ": " + "Sent Fire,
Time: " + returnCurrentTime()+ "\n");
                }
                catch (Exception ex){
                    ex.printStackTrace();
                }
            }
        });

        floodingOne.floodingGui.jButton.addActionListener(
        new ActionListener(){
            public void actionPerformed( ActionEvent e )
            {
                try {//Put Desire Action Here

floodingOne.setData(addressOne.getLocalHost().toString(),returnCurrentTime(),"M
essage");

                floodingOne.setStatusFromGui();
                po.sendMulticastPacket(floodingOne,
socket,addressOne);

floodingOne.floodingGui.directLinkTextArea.append(sendNumFlooding++ + ": " +
"Sent Flooding, Time: " + returnCurrentTime() + "\n");
                }
            }
        });
    }
}

```

```

        catch (Exception ex){
            ex.printStackTrace();
        }));

    hydRuptureOne.hrGui.jButton.addActionListener(
        new ActionListener(){
            public void actionPerformed( ActionEvent e )
            {
                try { //Put Desire Action Here

hydRuptureOne.setData(addressOne.getLocalHost().toString(),returnCurrentTime(),
"Message");

                                hydRuptureOne.setStatusFromGui();
                                po.sendMulticastPacket(hydRuptureOne,

socket,addressOne);

hydRuptureOne.hrGui.directLinkTextArea.append(sendNumHrup++ + ": " + "Sent Hyd
Rupture, Time: " + returnCurrentTime()+"\n");
                }
                catch (Exception ex){
                    ex.printStackTrace();
                }));

        airRuptureOne.arGui.jButton.addActionListener(
            new ActionListener(){
                public void actionPerformed( ActionEvent e )
                {
                    try { //Put Desire Action Here

airRuptureOne.setData(addressOne.getLocalHost().toString(),returnCurrentTime(),
"Message");

                                airRuptureOne.setStatusFromGui();
                                po.sendMulticastPacket(airRuptureOne,

socket,addressOne);

airRuptureOne.arGui.directLinkTextArea.append(sendNumAr++ + ": " + "Sent Air
Rupture, Time: " + returnCurrentTime()+"\n");
                    }
                    catch (Exception ex){
                        ex.printStackTrace();
                    }));

        hotRunOne.hrGui.jButton.addActionListener(
            new ActionListener(){
                public void actionPerformed( ActionEvent e )
                {
                    try { //Put Desire Action Here

hotRunOne.setData(addressOne.getLocalHost().toString(),returnCurrentTime(),"Mes
sage");

                                hotRunOne.setStatusFromGui();
                                po.sendMulticastPacket(hotRunOne,

socket,addressOne);

hotRunOne.hrGui.directLinkTextArea.append(sendNumHrun++ + ": " + "Sent Hot Run,
Time: " + returnCurrentTime()+"\n");
                    }
                    catch (Exception ex){
                        ex.printStackTrace();
                    }));

        fastLeakOne.flGui.jButton.addActionListener(
            new ActionListener(){
                public void actionPerformed( ActionEvent e )
                {
                    try { //Put Desire Action Here

```

```

fastLeakOne.setData(addressOne.getHost().toString(),returnCurrentTime(),"M
essage");
                                fastLeakOne.setStatusFromGui();
                                po.sendMulticastPacket(fastLeakOne,
socket,addressOne);

fastLeakOne.flGui.directLinkTextArea.append(sendNumFl++ + ": " + "Sent Fast
Leak, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }
                                }));

                                slowLeakOne.slGui.jButton.addActionListener(
                                new ActionListener(){
                                    public void actionPerformed( ActionEvent e )
                                    {
                                        try { //Put Desire Action Here

slowLeakOne.setData(addressOne.getHost().toString(),returnCurrentTime(),"M
essage");
                                slowLeakOne.setStatusFromGui();
                                po.sendMulticastPacket(slowLeakOne,
socket,addressOne);

slowLeakOne.slGui.directLinkTextArea.append(sendNumSl++ + ": " + "Sent Slow
Leak, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }
                                }));

                                stmRuptureOne.srGui.jButton.addActionListener(
                                new ActionListener(){
                                    public void actionPerformed( ActionEvent e )
                                    {
                                        try { //Put Desire Action Here

stmRuptureOne.setData(addressOne.getHost().toString(),returnCurrentTime(),
"Message");
                                stmRuptureOne.setStatusFromGui();
                                po.sendMulticastPacket(stmRuptureOne,
socket,addressOne);

stmRuptureOne.srGui.directLinkTextArea.append(sendNumSr++ + ": " + "Sent Stm
Rupture, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }
                                }));

                                rxScramOne.rsGui.jButton.addActionListener(
                                new ActionListener(){
                                    public void actionPerformed( ActionEvent e )
                                    {
                                        try { //Put Desire Action Here

rxScramOne.setData(addressOne.getHost().toString(),returnCurrentTime(),"Me
ssage");
                                rxScramOne.setStatusFromGui();
                                po.sendMulticastPacket(rxScramOne,
socket,addressOne);

rxScramOne.rsGui.directLinkTextArea.append(sendNumRscram++ + ": " + "Sent Rx
Scram, Time: " + returnCurrentTime()+"\n");

```



```

    }
    catch (Exception ex){
        ex.printStackTrace();
    }));

radSpillOne.rsGui.jButton.addActionListener(
    new ActionListener(){
        public void actionPerformed( ActionEvent e )
        {
            try { //Put Desire Action Here

radSpillOne.setData(addressOne.getHost().toString(),returnCurrentTime(),"M
essage");

                                radSpillOne.setStatusFromGui();
                                po.sendMulticastPacket(radSpillOne,
socket,addressOne);

radSpillOne.rsGui.directLinkTextArea.append(sendNumRspill++ + ": " + "Sent Rad
Spill, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }));

ss.ssGui.jButton.addActionListener(
    new ActionListener(){
        public void actionPerformed( ActionEvent e )
        {
            try { //Put Desire Action Here

ss.setData(addressTwo.getHost().toString(),returnCurrentTime(),"Message");
                                ss.setStatusFromGui();
                                po.sendMulticastPacket(ss, socket,addressTwo);
                                ss.ssGui.directLinkTextArea.append(sendNumSs++ +
": " + "Sent Ship Status, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }));

es.esGui.jButton.addActionListener(
    new ActionListener(){
        public void actionPerformed( ActionEvent e )
        {
            try { //Put Desire Action Here

es.setData(addressTwo.getHost().toString(),returnCurrentTime(),"Message");
                                es.setStatusFromGui();
                                po.sendMulticastPacket(es, socket,addressTwo);
                                es.esGui.directLinkTextArea.append(sendNumEs++ +
": " + "Sent Eng Status, Time: " + returnCurrentTime()+"\n");
                                }
                                catch (Exception ex){
                                    ex.printStackTrace();
                                }));

cr.crGui.jButton.addActionListener(
    new ActionListener(){
        public void actionPerformed( ActionEvent e )
        {
            try { //Put Desire Action Here

cr.setData(addressTwo.getHost().toString(),returnCurrentTime(),"Message");
                                cr.setStatusFromGui();
                                po.sendMulticastPacket(cr, socket,addressTwo);
                                cr.crGui.directLinkTextArea.append(sendNumCr++ +
": " + "Sent Compartment Rig, Time: " + returnCurrentTime()+"\n");

```

```

        }
        catch (Exception ex){
            ex.printStackTrace();
        }
    }

    sa.saGui.jButton.addActionListener(
        new ActionListener(){
            public void actionPerformed( ActionEvent e )
            {
                try { //Put Desire Action Here

sa.setData(addressTwo.getHost().toString(),returnCurrentTime(),"Message");
                sa.setStatusFromGui();
                po.sendMulticastPacket(sa, socket,addressTwo);
                sa.saGui.directLinkTextArea.append(sendNumSa++ +
": " + "Sent Ship Atmosphere, Time: " + returnCurrentTime()+"\n");
                }
                catch (Exception ex){
                    ex.printStackTrace();
                }
            }
        }

        catch(BindException be){
            System.out.println("Unable to Connect to Network, \nCheck Your
Network Connection and Restart");
            fireOne.fireGui.directLinkTextArea.append("Unable to Connect to
Network, \nCheck Your Network Connection \nand Restart");
        }
        catch(Exception e){
            e.printStackTrace();
            System.out.println("Exception occured in clientCommunicate");
        }
    }

}

/**
 * stop() - not implemented, but called if run as an applet in a browser
 */
public void stop() {
}

/**
 * Destroy the applet, cleanup of threads and of casualties and ship threads
 *
 * @return void
 */
public void destroy() {
    clientOne.destroy();
}

/**
 * Set the preferences used in the client interface
 *
 * @return void
 */
private void setGUIPreferences(){
    //Set Preferences For GUI
    //Level "this"
    this.setEnabled(true);
    this.setSize(new Dimension(380, 720));
    this.getContentPane().setLayout(borderLayout1);
}

```

```

//Level 1
casualtyTabPanel.setTabPlacement(JTabbedPane.LEFT);
casualtyTabPanel.setForeground(new java.awt.Color(59, 80, 153));
casualtyTabPanel.setMaximumSize(new Dimension(380, 200));
casualtyTabPanel.setMinimumSize(new Dimension(10, 10));
casualtyTabPanel.setPreferredSize(new Dimension(380, 100));
casualtyTabPanel.setFont(new Font("Dialog",Font.BOLD,12));
}

/**
 * add the gui components to the client interface
 *
 * @return void
 */
private void addGuiComponents(){

//Level this
this.getContentPane().add(casualtyTabPanel, BorderLayout.WEST);

//Level 1
casualtyTabPanel.add(fireOne.fireGui, "Fire");
casualtyTabPanel.add(floodingOne.floodingGui, "Flooding");
casualtyTabPanel.add(hydRuptureOne.hrGui, "Hyd Rupture");
casualtyTabPanel.add(airRuptureOne.arGui, "Air Rupture");
casualtyTabPanel.add(hotRunOne.hrGui, "Hot Run");
casualtyTabPanel.add(fastLeakOne.flGui, "Fast Leak");
casualtyTabPanel.add(slowLeakOne.slGui, "Slow Leak");
casualtyTabPanel.add(stmRuptureOne.srGui, "Stm Rupture");
casualtyTabPanel.add(rxScramOne.rsGui, "Rx Scram");
casualtyTabPanel.add(radSpillOne.rsGui, "Rad Spill");
casualtyTabPanel.add(ss.ssGui, "Ship");
casualtyTabPanel.add(es.esGui, "Engineering");
casualtyTabPanel.add(cr.crGui, "Rig Status");
casualtyTabPanel.add(sa.saGui, "Atmospheres");

}

/**
 * Utility method to parses the time from an entire date object
 *
 * @return String
 */
private String returnCurrentTime(){
int index=1;
long time = System.currentTimeMillis();
Date date = new Date(time);
StringTokenizer st = new StringTokenizer(date.toString());
String holdDate[] = new String[7];

while(st.hasMoreTokens()){
    holdDate[index] = st.nextToken();

    index++;
}
return holdDate[4];//The Current Time: 4th component of date object
}

/**Entry Point, used to intialize the applet as an application, allows
program to function as
 * an applet or an application
 *
 * @param args[] An array of methods from the command line arguments
 * @return void
 */
public static void main(String[] args) {

```

```

    Client applet = new Client(args);
    applet.isStandalone = true;
    JFrame frame = new JFrame();
    frame.setTitle(args[0] + ",Casualty Traffic " +args[1] +" Ship Traffic " +
args[2]);
    frame.getContentPane().add(applet, BorderLayout.CENTER);

    //Exit process on window close
    frame.addWindowListener(
        new WindowAdapter(){
            public void windowClosing( WindowEvent e )
            {
                System.exit(0);
            }
        }
    );
    applet.init();//initialize call
    applet.start();//start call

    try{
        Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        frame.setLocation((d.width - frame.getSize().width) / 2, (d.height -
frame.getSize().height) / 2);
        frame.setSize(380,720);
        frame.setVisible(true);
    }
    catch(Exception e){}
}
// static initializer for setting look & feel
static {
    try {
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception e) {}
}
}

```

Class: Server Object

```
package swipNet.control;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.border.*;
import java.net.*; //Socket objects
import java.util.*; //For Date
import swipNet.utility.*; //Post Office
import swipNet.dcObjects.*; //Fire, etc
import swipNet.shipObjects.*; //ShipStatus, etc
import swipNet.gui.*; //for control bar gui

/**
 * Server is a listener object, it subscribes to three multicast address and
 * receives
 * two casualty objects, and ship objects; All three run within 3 communicate
 * threads;
 * Server creates its on gui objects and uses Postoffice to receive sent
 * packets<p>
 * @author LT William G. Wilkins
 * @version 1.0
 */

public class Server extends JApplet {

    boolean isStandalone = false;
    //ImageIcon icon = new ImageIcon ("swipNet.control.duke.gif");

    public JSplitPanesplitPaneMiddle;
    public JSplitPanesplitPaneCaslandMiddle;
    public JSplitPanesplitPaneAllBottom;
    public JSplitPanesplitPaneAll;
    JLabel sub = new JLabel ( new ImageIcon("MySub.jpg"));
    JPanel subPanel = new JPanel();

    //Creation of GUI Objects

    /**Panel One- objects that hold the status of casualty one*/
    //Level 1
    protected JTabbedPane casualtyTabPaneA = new JTabbedPane();

    //Level 2
    Fire fireOne = new Fire(true); //set true since a server
    Flooding floodingOne = new Flooding(true);
    Hydrapture hydraptureOne = new Hydrapture(true);
    AirRapture airRaptureOne = new AirRapture(true);
    HotRun hotRunOne = new HotRun(true);
    FastLeak fastLeakOne = new FastLeak(true);
    SlowLeak slowLeakOne = new SlowLeak(true);
    StmRapture stmRaptureOne = new StmRapture(true);
    RxScram rxScramOne = new RxScram(true);
    RadSpill radSpillOne = new RadSpill(true);

    /**Panel Two- objects that hold the status of casualty two*/
    //Level 1
    protected JTabbedPane casualtyTabPaneB = new JTabbedPane();
    //Level 2
    Fire fireTwo = new Fire(true); //set true since a server
```

```

Flooding    floodingTwo = new Flooding(true);
HydRupture  hydRuptureTwo = new HydRupture(true);
AirRupture  airRuptureTwo = new AirRupture(true);
HotRun      hotRunTwo = new HotRun(true);
FastLeak    fastLeakTwo = new FastLeak(true);
SlowLeak    slowLeakTwo = new SlowLeak(true);
StmRupture  stmRuptureTwo = new StmRupture(true);
RxScram     rxScramTwo = new RxScram(true);
RadSpill    radSpillTwo = new RadSpill(true);

/**Panel Center- objects that hold the status of ship objects*/
//Level 1
protected JTabbedPane casualtyTabPaneC = new JTabbedPane();
//Level 2
ShipStatus ssThree = new ShipStatus(true); //set true since a server
EngineeringStatus esThree = new EngineeringStatus(true);
CompartmentRigs crThree = new CompartmentRigs(true);
ShipAtmospheres saThree = new ShipAtmospheres(true);

//Misc Creation
//JTabbedPane reportPane = new JTabbedPane(); //Holds report info
JPanel reportPanel = new JPanel(); //Holds report info
JTextArea reportTextArea = new JTextArea(250,180);
JScrollPane reportJSP = new JScrollPane(reportTextArea);

public ControlBarGui bar = new ControlBarGui();
BorderLayout BorderLayout1 = new BorderLayout();

CommunicateOne commOne;
CommunicateTwo commTwo;
CommunicateThree commThree;

Thread casualtyOne; //Threads for communication between objects
Thread casualtyTwo;
Thread shipThree;

/**
 * Server() constructor, used if no arguments are supplied from user
 */
public Server() {

    this.setName("Casualty Traffic on " + "228.7.5.4" + " and " + "228.7.5.5"
+ " Ship Traffic on " + "228.7.5.6");

    fireOne.setOwner("DCC", "228.7.5.4");
    fireOne.fireGui.host.setText("Listen on 228.7.5.4");
    fireTwo.setOwner("DCC", "228.7.5.5");
    fireTwo.fireGui.host.setText("Listen on 228.7.5.5");

    floodingOne.setOwner("DCC", "228.7.5.4");
    floodingOne.floodingGui.host.setText("Listen on 228.7.5.4");
    floodingTwo.setOwner("DCC", "228.7.5.5");
    floodingTwo.floodingGui.host.setText("Listen on 228.7.5.5");

    hydRuptureOne.setOwner("DCC", "228.7.5.4");
    hydRuptureOne.hrGui.host.setText("Listen on 228.7.5.4");
    hydRuptureTwo.setOwner("DCC", "228.7.5.5");
    hydRuptureTwo.hrGui.host.setText("Listen on 228.7.5.5");

    airRuptureOne.setOwner("DCC", "228.7.5.4");
    airRuptureOne.arGui.host.setText("Listen on 228.7.5.4");
    airRuptureTwo.setOwner("DCC", "228.7.5.5");
    airRuptureTwo.arGui.host.setText("Listen on 228.7.5.5");

    hotRunOne.setOwner("DCC", "228.7.5.4");

```

```

        hotRunOne.hrGui.host.setText("Listen on 228.7.5.4");
        hotRunTwo.setOwner("DCC", "Listen on 228.7.5.5");
        hotRunTwo.hrGui.host.setText("Listen on 228.7.5.5");

        fastLeakOne.setOwner("DCC", "228.7.5.4");
        fastLeakOne.flGui.host.setText("Listen on 228.7.5.4");
        fastLeakTwo.setOwner("DCC", "228.7.5.5");
        fastLeakTwo.flGui.host.setText("Listen on 228.7.5.5");

        slowLeakOne.setOwner("DCC", "228.7.5.4");
        slowLeakOne.slGui.host.setText("Listen on 228.7.5.4");
        slowLeakTwo.setOwner("DCC", "228.7.5.5");
        slowLeakTwo.slGui.host.setText("Listen on 228.7.5.5");

        stmRuptureOne.setOwner("DCC", "228.7.5.4");
        stmRuptureOne.srGui.host.setText("Listen on 228.7.5.4");
        stmRuptureTwo.setOwner("DCC", "228.7.5.5");
        stmRuptureTwo.srGui.host.setText("Listen on 228.7.5.5");

        rxScramOne.setOwner("DCC", "228.7.5.4");
        rxScramOne.rsGui.host.setText("Listen on 228.7.5.4");
        rxScramTwo.setOwner("DCC", "228.7.5.5");
        rxScramTwo.rsGui.host.setText("Listen on 228.7.5.5");

        radSpillOne.setOwner("DCC", "228.7.5.4");
        radSpillOne.rsGui.host.setText("Listen on 228.7.5.4");
        radSpillTwo.setOwner("DCC", "228.7.5.5");
        radSpillTwo.rsGui.host.setText("Listen on 228.7.5.5");

        ssThree.setOwner("DCC", "228.7.5.6");
        ssThree.ssGui.host.setText("DCC" + ", Listen on " + "228.7.5.6");
        esThree.setOwner("DCC", "228.7.5.6");
        esThree.esGui.host.setText("DCC" + ", Listen on " + "228.7.5.6");
        crThree.setOwner("DCC", "228.7.5.6");
        crThree.crGui.host.setText("DCC" + ", Listen on " + "228.7.5.6");
        saThree.setOwner("DCC", "228.7.5.6");
        saThree.saGui.host.setText("DCC" + ", Listen on " + "228.7.5.6");

        commOne = new CommunicateOne("228.7.5.4");
        commTwo = new CommunicateTwo("228.7.5.5");
        commThree= new CommunicateThree("228.7.5.6");
        casualtyOne = new Thread(commOne);
        casualtyTwo = new Thread(commTwo);
        shipThree = new Thread(commThree);
    }
    /**
     * Server constructor - new instance of server created with arguments
     *
     * @param args used to set gui name and owner if arguments are supplied from
     user
     */

    public Server(String [] args){
        this.setName(args[0] + ", Casualty Traffic on " + args[1] + " and " +
        args[2] + " Ship Traffic on " + args[3]);

        fireOne.setOwner(args[0],args[1] );
        fireOne.fireGui.host.setText(args[0] + ", Listen on " + args[1]);
        fireTwo.setOwner(args[0],args[2] );
        fireTwo.fireGui.host.setText(args[0] + ", Listen on " + args[2]);

        floodingOne.setOwner(args[0],args[1] );
        floodingOne.floodingGui.host.setText(args[0] + ", Listen on " +
args[1]);
        floodingTwo.setOwner(args[0],args[2]);
        floodingTwo.floodingGui.host.setText(args[0] + ", Listen on " +
args[2]);

```

```

hydRuptureOne.setOwner(args[0],args[1] );
hydRuptureOne.hrGui.host.setText(args[0] + ", Listen on " + args[1]);
hydRuptureTwo.setOwner(args[0],args[2]);
hydRuptureTwo.hrGui.host.setText(args[0] + ", Listen on " + args[2]);

airRuptureOne.setOwner(args[0],args[1] );
airRuptureOne.arGui.host.setText(args[0] + ", Listen on " + args[1]);
airRuptureTwo.setOwner(args[0],args[2]);
airRuptureTwo.arGui.host.setText(args[0] + ", Listen on " + args[2]);

hotRunOne.setOwner(args[0],args[1] );
hotRunOne.hrGui.host.setText(args[0] + ", Listen on " + args[1]);
hotRunTwo.setOwner(args[0],args[2]);
hotRunTwo.hrGui.host.setText(args[0] + ", Listen on " + args[2]);

fastLeakOne.setOwner(args[0],args[1] );
fastLeakOne.flGui.host.setText(args[0] + ", Listen on " + args[1]);
fastLeakTwo.setOwner(args[0],args[2]);
fastLeakTwo.flGui.host.setText(args[0] + ", Listen on " + args[2]);

slowLeakOne.setOwner(args[0],args[1] );
slowLeakOne.slGui.host.setText(args[0] + ", Listen on " + args[1]);
slowLeakTwo.setOwner(args[0],args[2]);
slowLeakTwo.slGui.host.setText(args[0] + ", Listen on " + args[2]);

stmRuptureOne.setOwner(args[0],args[1] );
stmRuptureOne.srGui.host.setText(args[0] + ", Listen on " + args[1]);
stmRuptureTwo.setOwner(args[0],args[2]);
stmRuptureTwo.srGui.host.setText(args[0] + ", Listen on " + args[2]);

rxScramOne.setOwner(args[0],args[1] );
rxScramOne.rsGui.host.setText(args[0] + ", Listen on " + args[1]);
rxScramTwo.setOwner(args[0],args[2]);
rxScramTwo.rsGui.host.setText(args[0] + ", Listen on " + args[2]);

radSpillOne.setOwner(args[0],args[1] );
radSpillOne.rsGui.host.setText(args[0] + ", Listen on " + args[1]);
radSpillTwo.setOwner(args[0],args[2]);
radSpillTwo.rsGui.host.setText(args[0] + ", Listen on " + args[2]);

ssThree.setOwner(args[0],args[3]);
ssThree.ssGui.host.setText(args[0] + ", Listen on " + args[3]);
esThree.setOwner(args[0],args[3]);
esThree.esGui.host.setText(args[0] + ", Listen on " + args[3]);
crThree.setOwner(args[0],args[3]);
crThree.crGui.host.setText(args[0] + ", Listen on " + args[3]);
saThree.setOwner(args[0],args[3]);
saThree.saGui.host.setText(args[0] + ", Listen on " + args[3]);

commOne = new CommunicateOne(args[1]);
commTwo = new CommunicateTwo(args[2]);
commThree= new CommunicateThree(args[3]);
casualtyOne = new Thread(commOne);
casualtyTwo = new Thread(commTwo);
shipThree = new Thread(commThree);
}

/**
 * init() - nest jbinit(), to catch exception and use Jbuilders gui builder
 interface
 */
public void init() {
    try {
        jbInit();
    }

```



```

    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

/**
 * jbinit(), used in Jbuilders gui builder interface, creates the GUI
 */
private void jbInit() throws Exception {
    this.setJMenuBar(bar);
    this.setGUIPreferences();
    this.addGuiComponents();
}

/**
 * start(), starts the casualty and ship threads
 */
public void start() {
    casualtyOne.start();
    casualtyTwo.start();
    shipThree.start();
}

/**
 * CommunicateOne - inner class that defines communication of casualty one
 */
private class CommunicateOne implements Runnable{

    int portNumber = 5000;
    MulticastSocket socket;
    InetAddress address;
    PostOffice po = new PostOffice();
    String multicastAddressCasualty= "228.7.5.4";//Default

    long time = System.currentTimeMillis();
    Date date = new Date(time);
    String threadName;

    //For Receive
    int reportNumber = 0;
    Object holdObject = new Object();
    String holdDataOneA[] = new String[6];
    String holdDataTwoA[] = new String[6];

    Fire fireReceive = new Fire(true);//used to temp hold the received objects
    Flooding floodingReceive = new Flooding(true);
    HydRupture hydRuptureReceive = new HydRupture(true);
    AirRupture airRuptureReceive = new AirRupture(true);
    HotRun hotRunReceive = new HotRun(true);
    FastLeak fastLeakReceive = new FastLeak(true);
    SlowLeak slowLeakReceive = new SlowLeak(true);
    StmRupture stmRuptureReceive = new StmRupture(true);
    RxScram rxScramReceive = new RxScram(true);
    RadSpill radSpillReceive = new RadSpill(true);

    /**
     * CommunicateOne() - contructor, with String input address
     *
     * @param mcAddress - allows to set multicast address for this thread
     */
    public CommunicateOne(String mcAddress){

```

```

        multicastAddressCasualty = mcAddress;
    }
    /**
    * Cycles through a while loop and receives multicast packets, determines its
    type, and updates the listener gui
    */
    public void run()
    {
        try
        {
            socket = new MulticastSocket(portNumber);
            address = InetAddress.getByName(multicastAddressCasualty);
            socket.joinGroup(address);
            threadName = "Casualty on " +
multicastAddressCasualty.toString();
            Thread.currentThread().setName(threadName);

            while(true){

                holdObject = po.receiveMulticastPacket(socket,address);

                reportNumber = reportNumber +1;
                if (holdObject instanceof Fire){
                    fireReceive = (Fire)holdObject;
                    holdDataOneA = fireReceive.getData();
                    holdDataTwoA = fireOne.getData();
                    reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
                    "\n      "+ holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
                    "\n      Time: " + holdDataOneA[4]
+"\"");
                    if (fireOne.fireGui.doUpdate == true){//If the pause
button is not pressed allow updates.
                        fireReceive.copyStatus(fireOne);//Copies only the
casualty specifics, not owner, etc.
                    }
                }
                if (holdObject instanceof Flooding){
                    floodingReceive = (Flooding)holdObject;
                    holdDataOneA = floodingReceive.getData();
                    holdDataTwoA = floodingOne.getData();
                    reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
                    "\n      "+ holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
                    "\n      Time: " + holdDataOneA[4]
+"\"");
                    floodingReceive.copyStatus(floodingOne);
                }
                if (holdObject instanceof HydRupture){
                    hydRuptureReceive = (HydRupture)holdObject;
                    holdDataOneA = hydRuptureReceive.getData();
                    holdDataTwoA = hydRuptureOne.getData();
                    reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
                    "\n      "+ holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
                    "\n      Time: " + holdDataOneA[4]
+"\"");
                    hydRuptureReceive.copyStatus(hydRuptureOne);
                }
                if (holdObject instanceof AirRupture){
                    airRuptureReceive = (AirRupture)holdObject;
                    holdDataOneA = airRuptureReceive.getData();

```

```

        holdDataTwoA = airRuptureOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
        "\n " + holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
        "\n Time: " + holdDataOneA[4]
+"\"");
        airRuptureReceive.copyStatus(airRuptureOne);
    }
    if (holdObject instanceof HotRun){
        hotRunReceive = (HotRun)holdObject;
        holdDataOneA = hotRunReceive.getData();
        holdDataTwoA = hotRunOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
        "\n " + holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
        "\n Time: " + holdDataOneA[4]
+"\"");
        hotRunReceive.copyStatus(hotRunOne);
    }
    if (holdObject instanceof FastLeak){
        fastLeakReceive = (FastLeak)holdObject;
        holdDataOneA = fastLeakReceive.getData();
        holdDataTwoA = fastLeakOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
        "\n " + holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
        "\n Time: " + holdDataOneA[4]
+"\"");
        fastLeakReceive.copyStatus(fastLeakOne);
    }
    if (holdObject instanceof SlowLeak){
        slowLeakReceive = (SlowLeak)holdObject;
        holdDataOneA = slowLeakReceive.getData();
        holdDataTwoA = slowLeakOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
        "\n " + holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
        "\n Time: " + holdDataOneA[4]
+"\"");
        slowLeakReceive.copyStatus(slowLeakOne);
    }
    if (holdObject instanceof StmRupture){
        stmRuptureReceive = (StmRupture)holdObject;
        holdDataOneA = stmRuptureReceive.getData();
        holdDataTwoA = stmRuptureOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+
        "\n " + holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
        "\n Time: " + holdDataOneA[4]
+"\"");
        stmRuptureReceive.copyStatus(stmRuptureOne);
    }
    if (holdObject instanceof RxScram){
        rxScramReceive = (RxScram)holdObject;
        holdDataOneA = rxScramReceive.getData();
        holdDataTwoA = rxScramOne.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\""+

```

```

Updated"+ " on " + holdDataOneA[2] +
                                "\n    "+ holdDataOneA[0] + "
                                "\n    Time: " + holdDataOneA[4]
+"\\n");
                                rxScramReceive.copyStatus(rxScramOne);
                                }
                                if (holdObject instanceof RadSpill){
                                radSpillReceive = (RadSpill)holdObject;
                                holdDataOneA = radSpillReceive.getData();
                                holdDataTwoA = radSpillOne.getData();
                                reportTextArea.append(reportNumber + ": "+ "\\n" +
holdDataTwoA[1]+ " ,this is " + holdDataOneA[1]+"\\n"+
                                "\n    "+ holdDataOneA[0] + "
Updated"+ " on " + holdDataOneA[2] +
                                "\n    Time: " + holdDataOneA[4]
+"\\n");
                                radSpillReceive.copyStatus(radSpillOne);
                                }

                                } //end While
                                } //end Try

                                catch(BindException be){
                                System.out.println("Unable to Connect to Network, \\nCheck Your
Network Connection and Restart");
                                reportTextArea.append("Unable to Connect to Network, \\nCheck Your
Network Connection \\nand Restart");
                                }
                                catch(Exception e){
                                e.printStackTrace();
                                System.out.println("Exception occured in commThreadOne");
                                }
                                }
                                }

/**
 * CommunicateTwo - inner class that defines communication of casualty Two
 */

private class CommunicateTwo implements Runnable{

    int portNumber = 5000;
    MulticastSocket socket;
    InetAddress address;
    PostOffice po = new PostOffice();
    String multicastAddressCasualty= "228.7.5.5";//Default

    long time = System.currentTimeMillis();
    Date date = new Date(time);
    String threadName;

    //For Receive
    int reportNumber = 0;
    Object holdObject = new Object();
    String holdDataOneB[] = new String[6];
    String holdDataTwoB[] = new String[6];

    Fire fireReceive = new Fire(true);
    Flooding floodingReceive = new Flooding(true);
    HydRupture hydRuptureReceive = new HydRupture(true);
    AirRupture airRuptureReceive = new AirRupture(true);
    HotRun hotRunReceive = new HotRun(true);
    FastLeak fastLeakReceive = new FastLeak(true);
    SlowLeak slowLeakReceive = new SlowLeak(true);
    StmRupture stmRuptureReceive = new StmRupture(true);
    RxScram rxScramReceive = new RxScram(true);

```

```

RadSpill    radSpillReceive = new RadSpill(true);

/**
 * CommunicateTwo() - constructor, with String input address
 *
 * @param mcAddress - allows to set multicast address for this thread
 */

public CommunicateTwo(String mcAddress){
    multicastAddressCasualty = mcAddress;
}

/**
 * Cycles through a while loop and receives multicast packets, determines its
 * type, and updates the listener gui
 */
public void run()
{
    try
    {
        socket = new MulticastSocket(portNumber);
        address = InetAddress.getByName(multicastAddressCasualty);
        socket.joinGroup(address);
        threadName = "Casualty on " +
multicastAddressCasualty.toString();
        Thread.currentThread().setName(threadName);

        while(true){

            holdObject = po.receiveMulticastPacket(socket,address);

            reportNumber = reportNumber +1;
            if (holdObject instanceof Fire){
                fireReceive = (Fire)holdObject;
                holdDataOneB = fireReceive.getData();
                holdDataTwoB = fireTwo.getData();

                reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                "\n    Time: " + holdDataOneB[4]
                +"\n");
                if (fireTwo.fireGui.doUpdate == true){//If the pause
button is not pressed allow updates.
                    fireReceive.copyStatus(fireTwo);//Copies only the
casualty specifics, not owner, etc.
                }
            }
            if (holdObject instanceof Flooding){
                floodingReceive = (Flooding)holdObject;
                holdDataOneB = floodingReceive.getData();
                holdDataTwoB = floodingTwo.getData();
                reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                "\n    Time: " + holdDataOneB[4]
                +"\n");
                floodingReceive.copyStatus(floodingTwo);
            }
        }
    }
}

```

```

        if (holdObject instanceof HydRupture){
            hydRuptureReceive = (HydRupture)holdObject;
            holdDataOneB = hydRuptureReceive.getData();
            holdDataTwoB = hydRuptureTwo.getData();
            reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                                "\n    Time: " + holdDataOneB[4]
+"\"");
            hydRuptureReceive.copyStatus(hydRuptureTwo);
        }

        if (holdObject instanceof AirRupture){
            airRuptureReceive = (AirRupture)holdObject;
            holdDataOneB = airRuptureReceive.getData();
            holdDataTwoB = airRuptureTwo.getData();
            reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                                "\n    Time: " + holdDataOneB[4]
+"\"");
            airRuptureReceive.copyStatus(airRuptureTwo);
        }

        if (holdObject instanceof HotRun){
            hotRunReceive = (HotRun)holdObject;
            holdDataOneB = hotRunReceive.getData();
            holdDataTwoB = hotRunTwo.getData();
            reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                                "\n    Time: " + holdDataOneB[4]
+"\"");
            hotRunReceive.copyStatus(hotRunTwo);
        }

        if (holdObject instanceof FastLeak){
            fastLeakReceive = (FastLeak)holdObject;
            holdDataOneB = fastLeakReceive.getData();
            holdDataTwoB = fastLeakTwo.getData();
            reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                                "\n    Time: " + holdDataOneB[4]
+"\"");
            fastLeakReceive.copyStatus(fastLeakTwo);
        }

        if (holdObject instanceof SlowLeak){
            slowLeakReceive = (SlowLeak)holdObject;
            holdDataOneB = slowLeakReceive.getData();
            holdDataTwoB = slowLeakTwo.getData();
            reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
                                "\n    "+ holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
                                "\n    Time: " + holdDataOneB[4]
+"\"");
            slowLeakReceive.copyStatus(slowLeakTwo);
        }

        if (holdObject instanceof StmRupture){
            stmRuptureReceive = (StmRupture)holdObject;
            holdDataOneB = stmRuptureReceive.getData();
            holdDataTwoB = stmRuptureTwo.getData();

```

```

        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
        "\n " + holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
        "\n Time: " + holdDataOneB[4]
+"\"n");
        stmRuptureReceive.copyStatus(stmRuptureTwo);
    }

    if (holdObject instanceof RxScram){
        rxScramReceive = (RxScram)holdObject;
        holdDataOneB = rxScramReceive.getData();
        holdDataTwoB = rxScramTwo.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
        "\n " + holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
        "\n Time: " + holdDataOneB[4]
+"\"n");
        rxScramReceive.copyStatus(rxScramTwo);
    }

    if (holdObject instanceof RadSpill){
        radSpillReceive = (RadSpill)holdObject;
        holdDataOneB = radSpillReceive.getData();
        holdDataTwoB = radSpillTwo.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoB[1]+ " ,this is " + holdDataOneB[1]+"\""+
        "\n " + holdDataOneB[0] + "
Updated"+ " on " + holdDataOneB[2] +
        "\n Time: " + holdDataOneB[4]
+"\"n");
        radSpillReceive.copyStatus(radSpillTwo);
    }
}

}

catch(Exception e){
    e.printStackTrace();
    System.out.println("Exception occured in commThreadTwo");
}
}

}

/**
 * CommunicateThree - inner class that defines communication of ship objects
 */
private class CommunicateThree implements Runnable{

    int portNumber = 5000;
    MulticastSocket socket;
    InetAddress address;
    PostOffice po = new PostOffice();
    String multicastAddressCasualty= "228.7.5.6";//Default
    long time = System.currentTimeMillis();
    Date date = new Date(time);
    String threadName;
    //For Receive
    int reportNumber = 0;
    Object holdObject = new Object();
    String holdDataOneC[] = new String[6];
    String holdDataTwoC[] = new String[6];
    ShipStatus ssReceive = new ShipStatus(true);
    EngineeringStatus esReceive = new EngineeringStatus(true);
    CompartmentRigs crReceive = new CompartmentRigs(true);
    ShipAtmospheres saReceive = new ShipAtmospheres(true);

```

```

/**
 * CommunicateOne() - constructor, with String input address
 *
 * @param mcAddress - allows to set multicast address for this thread
 */
public CommunicateThree(String mcAddress){
    multicastAddressCasualty = mcAddress;
}

/**
 * Cycles through a while loop and receives multicast packets, determines its
 * type, and updates the listener gui
 *
 * @return void
 */
public void run()
{
    try
    {
        socket = new MulticastSocket(portNumber);
        address = InetAddress.getByName(multicastAddressCasualty);
        socket.joinGroup(address);
        threadName = "Ship on " + multicastAddressCasualty.toString();
        Thread.currentThread().setName(threadName);

        while(true){

            holdObject = po.receiveMulticastPacket(socket,address);
            reportNumber = reportNumber +1;

            if (holdObject instanceof ShipStatus){
                ssReceive = (ShipStatus)holdObject;
                holdDataOneC = ssReceive.getData();
                holdDataTwoC = ssThree.getData();
                reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoC[1]+ " ,this is " + holdDataOneC[1]+"\""+
                                "\n      "+ holdDataOneC[0] + "
Updated"+ " on " + holdDataOneC[2] +
                                "\n    Time: " + holdDataOneC[4]
+"\"");
                if (ssThree.ssGui.doUpdate == true){//If the pause
button is not pressed allow updates.
                    ssReceive.copyStatus(ssThree);//Copies only the
casualty specifics, not owner, etc.
                }
            }
            if (holdObject instanceof EngineeringStatus){
                esReceive = (EngineeringStatus)holdObject;
                holdDataOneC = esReceive.getData();
                holdDataTwoC = esThree.getData();
                reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoC[1]+ " ,this is " + holdDataOneC[1]+"\""+
                                "\n      "+ holdDataOneC[0] + "
Updated"+ " on " + holdDataOneC[2] +
                                "\n    Time: " + holdDataOneC[4]
+"\"");
                esReceive.copyStatus(esThree);
            }
            if (holdObject instanceof CompartmentRigs){
                crReceive = (CompartmentRigs)holdObject;
                holdDataOneC = crReceive.getData();
            }
        }
    }
}

```



```

        holdDataTwoC = crThree.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoC[1]+ " ,this is " + holdDataOneC[1]+"\""+
        "\n " + holdDataOneC[0] + "
Updated"+ " on " + holdDataOneC[2] +
        "\n Time: " + holdDataOneC[4]
+"\"");
        crReceive.copyStatus(crThree);
    }
    if (holdObject instanceof ShipAtmospheres){
        saReceive = (ShipAtmospheres)holdObject;
        holdDataOneC = saReceive.getData();
        holdDataTwoC = saThree.getData();
        reportTextArea.append(reportNumber + ": " + "\"" +
holdDataTwoC[1]+ " ,this is " + holdDataOneC[1]+"\""+
        "\n " + holdDataOneC[0] + "
Updated"+ " on " + holdDataOneC[2] +
        "\n Time: " + holdDataOneC[4]
+"\"");
        saReceive.copyStatus(saThree);
    }

    }

    catch(Exception e){
        e.printStackTrace();
        System.out.println("Exception occured in shipThread");
    }

    }

    /**
     * stop() - not implemented, but called if run as an applet in a browser
     *
     * @return void
     */
    public void stop() {}

    /**
     * Destroy the applet, cleanup of threads and of casualties and ship threads
     * @return void
     */
    public void destroy() {
        casualtyOne.destroy();
        casualtyTwo.destroy();
        shipThree.destroy();
    }

    /**
     * Set the preferences used in the server gui
     *
     * @return void
     */
    private void setGUIPreferences(){

        //Set Preferences For GUI
        //Level "this"
        this.setEnabled(true);
        this.setName("Monitor Panel");
        this.setSize(new Dimension(1010, 775));
        this.getContentPane().setLayout(null);

```

```

        this.getContentPane().setBackground(new java.awt.Color(220,220,220)); //Very
Light Gray
        this.getContentPane().setVisible(true);

        reportPanel.setLayout( new BorderLayout() );
        reportPanel.setForeground(new java.awt.Color(59, 80, 153));
        reportPanel.setFont(new Font("Dialog",Font.BOLD,12));
        reportPanel.setMinimumSize(new Dimension(5, 5));

reportJSP.setHorizontalScrollBarPolicy(reportJSP.HORIZONTAL_SCROLLBAR_NEVER);

        subPanel.setForeground(new java.awt.Color(59, 80, 153));
        subPanel.setFont(new Font("Dialog",Font.BOLD,12));
        subPanel.setMinimumSize(new Dimension(0, 0));
        subPanel.setLayout( new BorderLayout() );

//Panel One
//Level 1
casualtyTabPaneA.setTabPlacement(JTabbedPane.LEFT);
casualtyTabPaneA.setForeground(new java.awt.Color(59, 80, 153));
casualtyTabPaneA.setFont(new Font("Dialog",Font.BOLD,12));
casualtyTabPaneA.setMinimumSize(new Dimension(5, 5));

//Panel Two
//Level 1
casualtyTabPaneB.setTabPlacement(JTabbedPane.RIGHT);
casualtyTabPaneB.setForeground(new java.awt.Color(59, 80, 153));
casualtyTabPaneB.setFont(new Font("Dialog",Font.BOLD,12));
casualtyTabPaneB.setMinimumSize(new Dimension(5, 5));

//Panel Center
//Level 1
casualtyTabPaneC.setTabPlacement(JTabbedPane.BOTTOM);
casualtyTabPaneC.setForeground(new java.awt.Color(59, 80, 153));
casualtyTabPaneC.setFont(new Font("Dialog",Font.BOLD,12));
casualtyTabPaneC.setMinimumSize(new Dimension(5, 5));
}

/**
 * add the gui components to the server interface
 *
 * @return void
 */
private void addGuiComponents(){

//Adding GUI components
//Level this

//Panel One
//Level 1

//casualtyTabPaneA.add(fireOne.fireGui, "Fire");
casualtyTabPaneA.add(fireOne.fireGui,"Fire");
casualtyTabPaneA.add(floodingOne.floodingGui, "Flooding");
casualtyTabPaneA.add(hydRuptureOne.hrGui, "Hyd Rupture");
casualtyTabPaneA.add(airRuptureOne.arGui, "Air Rupture");
casualtyTabPaneA.add(hotRunOne.hrGui, "Hot Run");
casualtyTabPaneA.add(fastLeakOne.flGui, "Fast Leak");
casualtyTabPaneA.add(slowLeakOne.slGui, "Slow Leak");
casualtyTabPaneA.add(stmRuptureOne.srGui, "Stm Rupture");

```

```

casualtyTabPaneA.add(rxScramOne.rsGui, "Rx Scram");
casualtyTabPaneA.add(radSpillOne.rsGui, "Rad Spill");

//Panel Two
//Level 1
casualtyTabPaneB.add(fireTwo.fireGui, "Fire");
casualtyTabPaneB.add(floodingTwo.floodingGui, "Flooding");
casualtyTabPaneB.add(hydRuptureTwo.hrGui, "Hyd Rupture");
casualtyTabPaneB.add(airRuptureTwo.arGui, "Air Rupture");
casualtyTabPaneB.add(hotRunTwo.hrGui, "Hot Run");
casualtyTabPaneB.add(fastLeakTwo.flGui, "Fast Leak");
casualtyTabPaneB.add(slowLeakTwo.slGui, "Slow Leak");
casualtyTabPaneB.add(stmRuptureTwo.srGui, "Stm Rupture");
casualtyTabPaneB.add(rxScramTwo.rsGui, "Rx Scram");
casualtyTabPaneB.add(radSpillTwo.rsGui, "Rad Spill");

//Panel Center
//Level 1
casualtyTabPaneC.add(ssThree.ssGui, "Ship");
casualtyTabPaneC.add(esThree.esGui, "Engineering");
casualtyTabPaneC.add(crThree.crGui, "Rig Status");
casualtyTabPaneC.add(saThree.saGui, "Atmospheres");

//SplitPanes
reportPanel.add( new JLabel( "          Station Reports and Ship Control
Messages" ), BorderLayout.NORTH );
reportPanel.add( reportJSP, BorderLayout.CENTER );

//subPanel.add(new JLabel( "SWIPNet" ), BorderLayout.NORTH);
subPanel.add(sub, BorderLayout.CENTER);

splitPaneMiddle = new JSplitPane(
JSplitPane.VERTICAL_SPLIT,true,reportPanel,casualtyTabPaneC );
splitPaneMiddle.setOneTouchExpandable(true);
splitPaneMiddle.setDividerSize(10);
splitPaneMiddle.setBorder(null);

splitPaneCaslandMiddle= new JSplitPane(
JSplitPane.HORIZONTAL_SPLIT,true,casualtyTabPaneA,splitPaneMiddle );
splitPaneCaslandMiddle.setOneTouchExpandable(true);
splitPaneCaslandMiddle.setDividerSize(10);
splitPaneCaslandMiddle.setBorder(null);

splitPaneAllBottom= new JSplitPane(
JSplitPane.HORIZONTAL_SPLIT,true,splitPaneCaslandMiddle,casualtyTabPaneB );
splitPaneAllBottom.setOneTouchExpandable(true);
splitPaneAllBottom.setDividerSize(10);
splitPaneAllBottom.setBorder(null);

splitPaneAll = new JSplitPane(
JSplitPane.VERTICAL_SPLIT,true,subPanel,splitPaneAllBottom );
splitPaneAll.setOneTouchExpandable(true);
splitPaneAll.setDividerSize(10);

splitPaneMiddle.setSize(1050,900);
splitPaneCaslandMiddle.setSize(1050,900);
splitPaneAllBottom.setSize(1050,900);
splitPaneAll.setSize(1050,900);

this.getContentPane().add( splitPaneAll);
splitPaneMiddle.setDividerLocation(0);
splitPaneCaslandMiddle.setDividerLocation(0);
splitPaneAllBottom.setDividerLocation(0);
splitPaneAll.setDividerLocation(0);
}

```

```

public void setDivider(){
    for (int ix = 0; ix <= 665; ix++){
        this.splitPaneAllBottom.setDividerLocation(ix);
        ix++;
        ix++;
        try {
            Thread.sleep(1);
        }
        catch(Exception e){
            System.out.println("Problem with Divider Setting");
        }
    }
    for (int ix = 0; ix <= 170; ix++){
        this.splitPaneMiddle.setDividerLocation(ix);
        ix++;
        ix++;
        try {
            Thread.sleep(1);
        }
        catch(Exception e){
            System.out.println("Problem with Divider Setting");
        }
    }

    for (int ix = 0; ix <= 365; ix++){
        this.splitPaneCaslandMiddle.setDividerLocation(ix);
        ix++;
        ix++;
        try {
            Thread.sleep(1);
        }
        catch(Exception e){
            System.out.println("Problem with Divider Setting");
        }
    }

    for (int ix = 0; ix <= 235; ix++){
        this.splitPaneAll.setDividerLocation(ix);
        ix++;
        try {
            Thread.sleep(1);
        }
        catch(Exception e){
            System.out.println("Problem with Divider Setting");
        }
    }
}
}

```

```

/**
 * Entry Point, used to intialize the applet as an application, allows
program to function as
 * an applet or an application
 *
 * @param args[] An array of methods from the command line arguments
 * @return void
 */
public static void main(String[] args) {
    Server applet = new Server(args);
    applet.isStandalone = true;
    JFrame frame = new JFrame();
}

```

```

        frame.setTitle(args[0] + ", Casualty Traffic on " + args[1] + " and " +
args[2] + " Ship Traffic on " + args[3]);
        frame.getContentPane().add(applet, BorderLayout.CENTER);

        //Exit process on window close
        frame.addWindowListener(
            new WindowAdapter(){
                public void windowClosing( WindowEvent e )
                {
                    System.exit(0);
                }
            }
        );

        applet.init();//initialize call
        applet.start();//start call

        try{
            Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
            frame.setLocation(5, 5);
            frame.setSize(1060,960);//990,720
            frame.setVisible(true);
        }
        catch(Exception e){

        }

        applet.setDivider();

    }
    // static initializer for setting look & feel
    static {
        try {
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName()
);
        }
        catch (Exception e) {}
    }

} //end server

```

Package: swipNet.dcObjects

Class: Fire

```
package swipNet.dcObjects;

import swipNet.gui.*;
import java.io.*;
import java.util.*;

/**
 * A Damage Control casualty object that holds information within strings that
 * can be directly sent over the network for Fire casualties; The fire object
 * is fully implemented and fully documented.<p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class Fire extends Object implements Serializable{

    /**Since network sends bytes you must know how to cast
     * the object when received on the other side. This number is read on
     * the other side , and casted appropriately
     */
    protected static int typeOfObject = 1;

    static int version = 1;
    static String type = new String ("FIRE");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    String dccCasLoc = new String();// Strings used to send the information
    String dccShipRigForFire = new String();
    String dccAtmWilimits = new String() ;

    String sceneMic = new String();
    String sceneAsstRqd = new String();
    String sceneInjuredPersonel = new String();
    String sceneDamageEquipment = new String();
    String sceneStatusOfFire = new String();

    String obaCrewMember = new String();
    String obaTimeRemaining = new String();

    String hoseRRHoses = new String();
    String hoseFRHoses = new String();

    /**
     *Fire class contains its own firegui for convienence and grouping, but it
     *is not actually sent on the Network.
     */
    public FireGui fireGui;

    /**
     * Constructor
     */
    public Fire() {}

    /**
     * This constructor allows the class to be used for Client or Server.
     */
}
```

```

    * Their GUIs are different; Fire class then passes it to its FireGui, that
    * actually uses it.
    * @param isServer if true, initialize as a Server, if not then display as
    * a Client
    */
    public Fire(boolean isServer) {
        fireGui = new FireGui(isServer);
    }

    /** This constructor allows casting to the appropriate object depending on
    * its type; The DataInputStream should be read in the SAME order that
    * toBytes places them onto the ByteArrayOutputStream, ORDER Counts.
    *
    * @param aBuffer[] a byte array used to read from
    */

    public Fire(byte aBuffer[])
    {

        ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
        DataInputStream dis = new DataInputStream(bis);

        try
        {
            typeOfObject = dis.readInt();
            version = dis.readInt();
            type = dis.readUTF();
            ownerName = dis.readUTF();
            multicastAddressOfOwner = dis.readUTF();
            machineName = dis.readUTF();
            timeSent = dis.readUTF();
            messageToSend = dis.readUTF();

            dccCasLoc = dis.readUTF();
            dccShipRigForFire = dis.readUTF();
            dccAtmWilimits= dis.readUTF();

            sceneMic= dis.readUTF();
            sceneAsstRqd= dis.readUTF();
            sceneInjuredPersonel= dis.readUTF();
            sceneDamageEquipment= dis.readUTF();
            sceneStatusOffire= dis.readUTF();

            obaCrewMember= dis.readUTF();
            obaTimeRemaining= dis.readUTF();

            hoseRRHoses= dis.readUTF();
            hoseFRHoses= dis.readUTF();
        }
        catch(Exception e)
        {
            System.out.println("Exception occured in Fire(byte[])");
        }
    }

    /**
    * This method converts or writes all instance variables within the fire
    * object to a stream to allow a byte buffer to be sent on the network,
    * again ORDER Counts
    * @return byte[] a byte array
    */

    public byte[] toBytes()
    {

```

```

ByteArrayOutputStream bos = new ByteArrayOutputStream();
DataOutputStream dos = new DataOutputStream(bos);

try
{
    dos.writeInt(typeOfObject);
    dos.writeInt(version);
    dos.writeUTF(type);

    dos.writeUTF(ownerName);
    dos.writeUTF(multicastAddressOfOwner);
    dos.writeUTF(machineName);
    dos.writeUTF(timeSent);
    dos.writeUTF(messageToSend);

    dos.writeUTF(dccCasLoc);
    dos.writeUTF(dccShipRigForFire);
    dos.writeUTF(dccAtmWiLimits);
    dos.writeUTF(sceneMic);
    dos.writeUTF(sceneAsstRqd);
    dos.writeUTF(sceneInjuredPersonel);
    dos.writeUTF(sceneDamageEquipment);
    dos.writeUTF(sceneStatusOfFire);
    dos.writeUTF(obaCrewMember);
    dos.writeUTF(obaTimeRemaining);
    dos.writeUTF(hoseRRHoses);
    dos.writeUTF(hoseFRHoses);
}
catch(IOException ioe)
{
    System.out.println("IOException - Unable to convert Fire Object to
Bytes");
    return null;
}

return bos.toByteArray();
}

/**
 * Retreives the generic ownership data of each fire object
 *
 * @return holdString a String array with each value filled
 */
public String[] getData(){

    String holdString[] = new String[6];

    holdString[0] = type;
    holdString[1] = ownerName;
    holdString[2] = multicastAddressOfOwner;
    holdString[3] = machineName;
    holdString[4] = timeSent;
    holdString[5] = messageToSend;

    return holdString;
}

/**
 * Takes the Strings from the callers Fire object and copies to its matching
 * text field
 * @param fire the object that you want to fill with received fire data
 *
 * @return void
 */
public void copyStatus(Fire fire){

```



```

        fire.fireGui.dccCasLocTF.setText(dccCasLoc);
        fire.fireGui.dccShipRigForFireTF.setText(dccShipRigForFire);
        fire.fireGui.dccAtmWiLimitsTF.setText(dccAtmWiLimits);
        fire.fireGui.sceneMicTF.setText(sceneMic);
        fire.fireGui.sceneAsstRqdTF.setText(sceneAsstRqd);
        fire.fireGui.sceneInjuredPersonelTF.setText(sceneInjuredPersonel);
        fire.fireGui.sceneDamageEquipmentTF.setText(sceneDamageEquipment);
        fire.fireGui.sceneStatusOfFireTF.setText(sceneStatusOfFire);
        fire.fireGui.obaCrewMemberTF.setText(obaCrewMember);
        fire.fireGui.obaTimeRemainingTF.setText(obaTimeRemaining);
        fire.fireGui.hoseRRHosesTF.setText(hoseRRHoses);
        fire.fireGui.hoseFRHosesTF.setText(hoseFRHoses);
        fire.fireGui.dccCasLocTF.setText(dccCasLoc);
        fire.fireGui.dccCasLocTF.setText(dccCasLoc);
        fire.fireGui.dccCasLocTF.setText(dccCasLoc);
    }

    /**
     * Set the ownerName and multicast address
     *
     * @param A ownerName, like DCC
     * @param B mulitcast you are sending on
     */
    public void setOwner (String A, String B){
        ownerName = A;
        multicastAddressOfOwner = B;
    }

    /**
     * Set other pertinent object data
     *
     * @param A IP address and domain name of the computer thats using object
     * @param B timeSent stamp
     * @param C any message to add during send
     * @return void
     */
    public void setData (String A, String B, String C){
        machineName = A;
        timeSent = B;
        messageToSend = C;
    }

    /**
     *When called takes the current Combo Box settings and copies it to its
     *corresponding String.
     *
     * @return void
     */
    public void setStatusFromGui(){
        dccCasLoc = (String)fireGui.dccCasLocCB.getSelectedItemAt();
        dccShipRigForFire =
        (String)fireGui.dccShipRigForFireCB.getSelectedItemAt();
        dccAtmWiLimits = (String)fireGui.dccAtmWiLimitsCB.getSelectedItemAt();
        sceneMic = (String)fireGui.sceneMicCB.getSelectedItemAt();
        sceneAsstRqd = (String)fireGui.sceneAsstRqdCB.getSelectedItemAt();
        sceneInjuredPersonel =
        (String)fireGui.sceneInjuredPersonelCB.getSelectedItemAt();
    }

```

```
        sceneDamageEquipment =  
(String)fireGui.sceneDamageEquipmentCB.getSelectedItem();  
sceneStatusOfFire=(String)fireGui.sceneStatusOfFireCB.getSelectedItem();  
        obaCrewMember = (String)fireGui.obaCrewMemberCB.getSelectedItem();  
        obaTimeRemaining =  
(String)fireGui.obaTimeRemainingCB.getSelectedItem();  
        hoseRRHoses = (String)fireGui.hoseRRHosesCB.getSelectedItem();  
        hoseFRHoses = (String)fireGui.hoseFRHosesCB.getSelectedItem();  
    }  
  
} //End Fire Class
```

Package: swipNet.gui

Class: FireGui

```
package swipNet.gui;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import swipNet.utility.Initialize;

/**
 * Graphical user interface(gui) that shows either Server or Client components
 * and represents the data contained within it matching dc or ship object;
 * gui isolated by itself to allow easier editing<p>
 * @author LT William G. Wilkins
 * @version 1.0
 */
public class FireGui extends JPanel {
    BorderLayout BorderLayout1 = new BorderLayout();
    /**
     * Create a Server interface(if true) and Client iinterface (if false)
     */
    protected boolean isServer = true;
    /**
     * used with toggle button to pause the listener so the interface can be
     read without continual updating
     */
    public boolean doUpdate = true;

    /**
     * provides set of Strings used to initialize Combo Boxes
     */
    Initialize initCB = new Initialize();

    //New Gui Components - Client
    //Level 1
    public JTabbedPane innerTabPane = new JTabbedPane();
    /**
     * For the Client - status displays
     */
    public JTextArea directLinkTextArea = new JTextArea(250,180);
    public JScrollPane dLinkJSP= new JScrollPane(directLinkTextArea);

    /**
     * For the Client - sned button
     */
    public JButton jButton = new JButton();
    /**
     * For the Server/Listener - pause feature
     */
    public JToggleButton jtButton = new JToggleButton();
    /**
     * Identification of Name and multicast channel listening on
     */
    public JTextField host = new JTextField();
    //Level 2
    public JPanel jDCCPanel = new JPanel();
    public JPanel jScenePanel = new JPanel();
    public JPanel jHosePanel = new JPanel();
    public JPanel jOBAPanel = new JPanel();

    //Level 3
    public JLabel clLabel1 = new JLabel();
```

```

    public JLabel srLabel2 = new JLabel();
    public JLabel atmLabel3 = new JLabel();
    public JLabel micLabel4 = new JLabel();
    public JLabel asstLabel5 = new JLabel();
    public JLabel ipLabel6 = new JLabel();
    public JLabel deLabel7 = new JLabel();
    public JLabel sofLabel8 = new JLabel();
    public JLabel cmLabel9 = new JLabel();
    public JLabel trLabel10 = new JLabel();
    public JLabel rrLabel11 = new JLabel();
    public JLabel frLabel12 = new JLabel();

    //For Client GUI and Populate from Initialize Class
    public JComboBox dccCasLocCB = new JComboBox(initCB.dccCasLocArray);
    public JComboBox dccShipRigForFireCB = new
JComboBox(initCB.dccShipRigForFireArray);
    public JComboBox dccAtmWiLimitsCB = new
JComboBox(initCB.dccAtmWiLimitsArray);
    public JComboBox sceneMicCB = new JComboBox(initCB.sceneMicArray);
    public JComboBox sceneAsstRqdCB = new JComboBox(initCB.sceneAsstRqdArray);
    public JComboBox sceneInjuredPersonelCB = new
JComboBox(initCB.sceneInjuredPersonelArray);
    public JComboBox sceneDamageEquipmentCB = new
JComboBox(initCB.sceneDamageEquipmentArray);
    public JComboBox sceneStatusOfFireCB = new
JComboBox(initCB.sceneStatusOfFireArray);
    public JComboBox obaCrewMemberCB = new
JComboBox(initCB.obaCrewMemberArray);
    public JComboBox obaTimeRemainingCB = new
JComboBox(initCB.obaTimeRemainingArray);
    public JComboBox hoseRRHosesCB = new JComboBox(initCB.hoseRRHosesArray);
    public JComboBox hoseFRHosesCB = new JComboBox(initCB.hoseFRHosesArray);

    //For Server GUI
    public JTextField dccCasLocTF = new JTextField();
    public JTextField dccShipRigForFireTF = new JTextField();
    public JTextField dccAtmWiLimitsTF = new JTextField();
    public JTextField sceneMicTF = new JTextField();
    public JTextField sceneAsstRqdTF = new JTextField();
    public JTextField sceneInjuredPersonelTF = new JTextField();
    public JTextField sceneDamageEquipmentTF = new JTextField();
    public JTextField sceneStatusOfFireTF = new JTextField();
    public JTextField obaCrewMemberTF = new JTextField();
    public JTextField obaTimeRemainingTF = new JTextField();
    public JTextField hoseRRHosesTF = new JTextField();
    public JTextField hoseFRHosesTF = new JTextField();

    /**
     * Constructor - jbInit() nested, within try{}
     */
    public FireGui() {
        try {
            jbInit();
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * Fire() Constructor, allows this class to be used for Client or Server;
     * Their GUIs are different.
     * @param isServer if true, initialize as a Server, if not then display as a
     Client

```

```

    */
    public FireGui(boolean isServer) {
        try {
            this.isServer = isServer;
            jbInit();
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * jbInit() used within JBuilder, to allow GUI Designing
     *
     * @return void
     */
    void jbInit() throws Exception {
        this.setLayout(borderLayout1);

        //Set GUI Preferences
        //Level Top 1
        this.setFont(new java.awt.Font("Dialog", 1, 12));
        this.setAlignmentX((float) 0.0);
        this.setAlignmentY((float) 5.0);
        this.setLayout(null);

        //Level 2
        innerTabPane.setBackground(Color.lightGray);
        innerTabPane.setFont(new java.awt.Font("Dialog", 1, 12));
        innerTabPane.setAlignmentY((float) 2.0);
        innerTabPane.setPreferredSize(new Dimension(220, 180));
        innerTabPane.setRequestFocusEnabled(false);
        innerTabPane.setBounds(new Rectangle(10, 13, 252, 442));

        dLinkJSP.setBounds(new Rectangle(10, 500, 260, 160));

        host.setBounds(new Rectangle(7, 460, 180, 30));
        jButton.setBounds(new Rectangle(190, 460, 80, 30));
        jButton.setBounds(new Rectangle(190, 460, 80, 30));

        //Level 3
        jDCCPanel.setLayout(null);
        jDCCPanel.setBackground(new java.awt.Color(221, 255, 235));
        jDCCPanel.setPreferredSize(new Dimension(250, 190));
        jScenePanel.setLayout(null);
        jScenePanel.setBackground(new java.awt.Color(192, 192, 239));
        jHosePanel.setLayout(null);
        jHosePanel.setBackground(new java.awt.Color(255, 223, 192));
        jHosePanel.setFont(new java.awt.Font("Dialog", 0, 9));
        jOBAPanel.setLayout(null);
        jOBAPanel.setBackground(new java.awt.Color(192, 175, 191));

        //Level 4
        clLabel1.setText("Casualty Location");
        clLabel1.setBounds(new Rectangle(19, 29, 121, 23));
        srLabel2.setText("Ship Rigged for Fire?");
        srLabel2.setBounds(new Rectangle(18, 91, 125, 17));
        atmLabel3.setToolTipText("");
        atmLabel3.setText("Atmosphere within Limits of:");
        atmLabel3.setBounds(new Rectangle(18, 151, 168, 14));
        micLabel4.setText("Man In Charge at Scene is:");
        micLabel4.setBounds(new Rectangle(15, 38, 170, 17));
    }

```

```

asstLabel5.setText("Scene Assistance Rqd?");
asstLabel5.setBounds(new Rectangle(16, 103, 159, 15));
ipLabel6.setText("Injured Personnel?");
ipLabel6.setBounds(new Rectangle(16, 164, 155, 17));
deLabel7.setText("Damaged Equipment?");
deLabel7.setBounds(new Rectangle(17, 231, 154, 15));
sofLabel8.setText("Status of Fire is:");
sofLabel8.setBounds(new Rectangle(17, 298, 151, 15));
cmLabel9.setText("OBA Crew Member:");
cmLabel9.setBounds(new Rectangle(11, 60, 137, 18));
trLabel10.setText("Time Remaining:");
trLabel10.setBounds(new Rectangle(81, 130, 151, 17));
rrLabel11.setText("Rapid Reponse Hose Team:");
rrLabel11.setBounds(new Rectangle(13, 56, 165, 19));
frLabel12.setText("Fast Response Hose Team:");
frLabel12.setBounds(new Rectangle(14, 133, 157, 16));

//For Client GUI
dccCasLocCB.setEditable(true); //Allows edit contents
dccShipRigForFireCB.setEditable(true);
dccAtmWilimitsCB.setEditable(true);
sceneMicCB.setEditable(true);
sceneAsstRqdCB.setEditable(true);
sceneInjuredPersonelCB.setEditable(true);
sceneDamageEquipmentCB.setEditable(true);
sceneStatusOffireCB.setEditable(true);
obaCrewMemberCB.setEditable(true);
obaTimeRemainingCB.setEditable(true);
hoseRRHosesCB.setEditable(true);
hoseFRHosesCB.setEditable(true);
dccCasLocCB.setBounds(new Rectangle(19, 55, 212, 26));
dccShipRigForFireCB.setBounds(new Rectangle(19, 114, 211, 27));
dccAtmWilimitsCB.setBounds(new Rectangle(19, 173, 212, 30));
sceneMicCB.setBounds(new Rectangle(14, 59, 222, 30));
sceneAsstRqdCB.setBounds(new Rectangle(14, 121, 220, 33));
sceneInjuredPersonelCB.setBounds(new Rectangle(14, 187, 219, 32));
sceneDamageEquipmentCB.setBounds(new Rectangle(15, 250, 216, 36));
sceneStatusOffireCB.setBounds(new Rectangle(15, 316, 215, 34));
obaCrewMemberCB.setBounds(new Rectangle(11, 81, 228, 34));
obaTimeRemainingCB.setBounds(new Rectangle(78, 151, 158, 35));
hoseRRHosesCB.setBounds(new Rectangle(13, 84, 212, 36));
hoseFRHosesCB.setBounds(new Rectangle(14, 153, 210, 34));

//For Server GUI
dccCasLocTF.setBounds(new Rectangle(19, 55, 212, 26));
dccShipRigForFireTF.setBounds(new Rectangle(19, 114, 211, 27));
dccAtmWilimitsTF.setBounds(new Rectangle(19, 173, 212, 30));
sceneMicTF.setBounds(new Rectangle(14, 59, 222, 30));
sceneAsstRqdTF.setBounds(new Rectangle(14, 121, 220, 33));
sceneInjuredPersonelTF.setBounds(new Rectangle(14, 187, 219, 32));
sceneDamageEquipmentTF.setBounds(new Rectangle(15, 250, 216, 36));
sceneStatusOffireTF.setBounds(new Rectangle(15, 316, 215, 34));
obaCrewMemberTF.setBounds(new Rectangle(11, 81, 228, 34));
obaTimeRemainingTF.setBounds(new Rectangle(78, 151, 158, 35));
hoseRRHosesTF.setBounds(new Rectangle(13, 84, 212, 36));
hoseFRHosesTF.setBounds(new Rectangle(14, 153, 210, 34));

//Add GUI Components
//Level 2
this.add(innerTabPane, null);

//this.add(jButton, null);
this.add(host, null);

//Level 3
innerTabPane.add(jDCCPanel, "DCC");

```

```

innerTabPane.add(jScenePanel, "Scene");
innerTabPane.add(jOBAPanel, "OBA");
innerTabPane.add(jHosePanel, "HOSE");

//Level 4
jDCCPanel.add(clLabel1, null);
jDCCPanel.add(srLabel2, null);
jDCCPanel.add(atmLabel3, null);
jScenePanel.add(micLabel4, null);
jScenePanel.add(sofLabel8, null);
jScenePanel.add(deLabel7, null);
jScenePanel.add(asstLabel5, null);
jScenePanel.add(ipLabel6, null);
jOBAPanel.add(trLabel10, null);
jOBAPanel.add(cmLabel9, null);
jHosePanel.add(rrLabel11, null);
jHosePanel.add(frLabel12, null);

/**
 * Creates different guis depending if a Server or Client
 */
if (isServer == true) {
    //For Server GUI
    jButton.setText("Pause");

    this.add(jButton, null);

    jDCCPanel.add(dccCasLocTF, null);
    jDCCPanel.add(dccShipRigForFireTF, null);
    jDCCPanel.add(dccAtmWiLimitsTF, null);
    jScenePanel.add(sceneMicTF, null);
    jScenePanel.add(sceneAsstRqdTF, null);
    jScenePanel.add(sceneInjuredPersonelTF, null);
    jScenePanel.add(sceneDamageEquipmentTF, null);
    jScenePanel.add(sceneStatusOffFireTF, null);
    jOBAPanel.add(obaCrewMemberTF, null);
    jOBAPanel.add(obaTimeRemainingTF, null);
    jHosePanel.add(hoseRRHosesTF, null);
    jHosePanel.add(hoseFRHosesTF, null);

    jButton.addItemListener(//Listener for the Pause feature
        new ItemListener() {
            public void itemStateChanged(ItemEvent e)
            {
                try { //Put Desire Action Here

                    if (e.getItemSelectable()==jButton){

                        if (e.getStateChange() ==
ItemEvent.SELECTED){

                            doUpdate = false;
                            jButton.setText("unPause");

                        }
                        else{
                            doUpdate =true;
                            jButton.setText("Pause");
                        }
                    } //end else
                } //end outer if
            } //end try
            catch (Exception ex){
                ex.printStackTrace();
            }
        }
    ); //end inner class and method call

```


Package: swipNet.shipObjects

Class: ShipStatus

```
package swipNet.shipObjects;

import java.io.*;
import java.util.*;
import swipNet.gui.*;

/**
 * A ship object that holds information within strings that can
 * be directly sent over the network for ShipAtmospheres parameters<p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class ShipStatus extends Object implements Serializable{

    static int typeOfObject = 20;
    static int version = 1;
    static String type = new String ("SHIPSTATUS");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");//Fix With
Function Call
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    String j = new String("Test");

    public ShipStatusGui ssGui;

    /**ShipStatus Constructor
     *
     */
    public ShipStatus() {}

    public ShipStatus(boolean isServer) {
        ssGui = new ShipStatusGui(isServer);
    }

    /** This constructor allows casting to the appropriate object depending on
     * its type. It should read in the same order that the toBytes places
     * onto the ByteArrayOutputStream.
     */
    public ShipStatus(byte aBuffer[])
    {

        ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
        DataInputStream dis = new DataInputStream(bis);

        try
        {
```

```

        typeOfObject = dis.readInt();
        version = dis.readInt();
        type = dis.readUTF();
        ownerName = dis.readUTF();
        multicastAddressOfOwner = dis.readUTF();
        machineName = dis.readUTF();
        timeSent = dis.readUTF();
        messageToSend = dis.readUTF();
        j = dis.readUTF();
    }
    catch(Exception e)
    {
        System.out.println("Exception occurred in (byte[])");
    }
}

/** toBytes method converts all instance variables within the object to a
 * stream to allow a byte buffer to be sent on the network
 */

public byte[] toBytes()
{
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(bos);

    try
    {
        dos.writeInt(typeOfObject);
        dos.writeInt(version);
        dos.writeUTF(type);

        dos.writeUTF(ownerName);
        dos.writeUTF(multicastAddressOfOwner);
        dos.writeUTF(machineName);
        dos.writeUTF(timeSent);
        dos.writeUTF(messageToSend);

        dos.writeUTF(j);
    }
    catch(IOException ioe)
    {
        System.out.println("IOException - Unable to convert Fire Object to
Bytes");
        return null;
    }

    return bos.toByteArray();
}

public String[] getData(){
    String holdString[] = new String[6];

    holdString[0] = type;
    holdString[1] = ownerName;
    holdString[2] = multicastAddressOfOwner;
    holdString[3] = machineName;
    holdString[4] = timeSent;
    holdString[5] = messageToSend;
}

```

```

        return holdString;
    }

    public void copyStatus(ShipStatus ss){
        ss.ssGui.jTFcurrentOp.setText(j);
    }

    public void setOwner (String A, String B){
        ownerName = A;
        multicastAddressOfOwner = B;
    }

    public void setData (String A, String B, String C){
        machineName = A;
        timeSent = B;
        messageToSend = C;
    }

    public void setStatusFromGui(){
        j = (String)ssGui.jCBcurrentOp.getSelectedItem();
    }
}

```

Class: EngineeringStatus

```
package swipNet.shipObjects;

import java.io.*;
import java.util.*;
import swipNet.gui.*;

/**
 * A ship object that holds information within strings that can
 * be directly sent over the network for EngineeringStatus parameters<p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class EngineeringStatus extends Object implements Serializable{

    static int typeOfObject = 21;
    static int version = 1;
    static String type = new String ("ENGINEERINGSTATUS");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");//Fix With
Function Call
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    String jRxPower = new String();
    String jPumpLineup = new String();
    String jLoopLineup = new String();

    String jSteamPower = new String();
    String jCurrentBell = new String();
    String jMaxBell = new String();

    String jElectricLineup = new String();
    String jDischargeRate = new String();
    String jAmpsRemain = new String();

    public EngineeringStatusGui esGui;

    /**Fire Constructor
     *
     */
    public EngineeringStatus() {}

    public EngineeringStatus(boolean isServer) {
        esGui = new EngineeringStatusGui(isServer);
    }

    /** This constructor allows casting to the appropriate object depending on
the
     * its type. It should read in the same order that the toBytes places
     * onto the ByteArrayOutputStream.
```

```

    */

    public EngineeringStatus(byte aBuffer[])
    {
        ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
        DataInputStream dis = new DataInputStream(bis);

        try
        {
            typeOfObject = dis.readInt();
            version = dis.readInt();
            type = dis.readUTF();
            ownerName = dis.readUTF();
            multicastAddressOfOwner = dis.readUTF();
            machineName = dis.readUTF();
            timeSent = dis.readUTF();
            messageToSend = dis.readUTF();

            jRxPower = dis.readUTF();
            jPumpLineup = dis.readUTF();
            jLoopLineup = dis.readUTF();

            jSteamPower = dis.readUTF();
            jCurrentBell = dis.readUTF();
            jMaxBell = dis.readUTF();

            jElectricLineup = dis.readUTF();
            jDischargeRate = dis.readUTF();
            jAmpsRemain = dis.readUTF();
        }
        catch(Exception e)
        {
            System.out.println("Exception occurred in (byte[])");
        }
    }

    /** toBytes method converts all instance variables within the object to a
     * stream to allow a byte buffer to be sent on the network
     */

    public byte[] toBytes()
    {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(bos);

        try
        {
            dos.writeInt(typeOfObject);
            dos.writeInt(version);
            dos.writeUTF(type);

            dos.writeUTF(ownerName);
            dos.writeUTF(multicastAddressOfOwner);
            dos.writeUTF(machineName);
            dos.writeUTF(timeSent);
            dos.writeUTF(messageToSend);

            dos.writeUTF(jRxPower);
            dos.writeUTF(jPumpLineup);
            dos.writeUTF(jLoopLineup);
        }
    }

```

```

        dos.writeUTF(jSteamPower);
        dos.writeUTF(jCurrentBell);
        dos.writeUTF(jMaxBell);

        dos.writeUTF(jElectricLineup);
        dos.writeUTF(jDischargeRate);
        dos.writeUTF(jAmpsRemain);

    }
    catch(IOException ioe)
    {
        System.out.println("IOException - Unable to convert Fire Object to
Bytes");
        return null;
    }

    return bos.toByteArray();
}

public String[] getData(){

    String holdString[] = new String[6];

    holdString[0] = type;
    holdString[1] = ownerName;
    holdString[2] = multicastAddressOfOwner;
    holdString[3] = machineName;
    holdString[4] = timeSent;
    holdString[5] = messageToSend;

    return holdString;

}

public void copyStatus(EngineeringStatus es){

    es.esGui.jRxPowerTF.setText(jRxPower);
    es.esGui.jPumpLineupTF.setText(jPumpLineup);
    es.esGui.jLoopLineupTF.setText(jLoopLineup);

    es.esGui.jSteamPowerTF.setText(jSteamPower);
    es.esGui.jCurrentBellTF.setText(jCurrentBell);
    es.esGui.jMaxBellTF.setText(jMaxBell);

    es.esGui.jElectricLineupTF.setText(jElectricLineup);
    es.esGui.jDischargeRateTF.setText(jDischargeRate);
    es.esGui.jAmpsRemainTF.setText(jAmpsRemain);

}

public void setOwner (String A, String B){

    ownerName = A;
    multicastAddressOfOwner = B;

}

public void setData (String A, String B, String C){

```

```

        machineName = A;
        timeSent = B;
        messageToSend = C;
    }

    public void setStatusFromGui(){

        jRxPower = (String)esGui.jRxPowerCB.getSelectedItem();
        jPumpLineup = (String)esGui.jPumpLineupCB.getSelectedItem();
        jLoopLineup = (String)esGui.jLoopLineupCB.getSelectedItem();

        jSteamPower = (String)esGui.jSteamPowerCB.getSelectedItem();
        jCurrentBell = (String)esGui.jCurrentBellCB.getSelectedItem();
        jMaxBell = (String)esGui.jMaxBellCB.getSelectedItem();

        jElectricLineup = (String)esGui.jElectricLineupCB.getSelectedItem();
        jDischargeRate = (String)esGui.jDischargeRateCB.getSelectedItem();
        jAmpsRemain = (String)esGui.jAmpsRemainCB.getSelectedItem();

    }

}

```

Class: CompartmentRigs

```
package swipNet.shipObjects;

import java.io.*;
import java.util.*;
import swipNet.gui.*;

/**
 * A ship object that holds information within strings that can
 * be directly sent over the network for CompartmentRigs parameters<p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class CompartmentRigs extends Object implements Serializable{

    static int typeOfObject = 22;
    static int version = 1;
    static String type = new String ("COMPARTMENTRIGS");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");//Fix With
Function Call
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    String jMainString1 = new String();String jMainString2 = new
String();String jMainString3 = new String();
    String jMainString4 = new String();String jMainString5 = new
String();String jMainString6 = new String();
    String jMainString7 = new String();

    String jFireString1 = new String (); String jFireString2 = new String
();String jFireString3 = new String ();
    String jFireString4 = new String ();String jFireString5 = new String
();String jFireString6 = new String ();
    String jFireString7 = new String ();String jFireString8 = new String
();String jFireString9 = new String ();
    String jFireString10 = new String ();String jFireString11 = new String
();String jFireString12 = new String ();
    String jFireString13 = new String ();String jFireString14 = new String ();

    String jFloodingString1 = new String ();String jFloodingString2 = new
String ();String jFloodingString3 = new String ();
    String jFloodingString4 = new String ();String jFloodingString5 = new
String ();String jFloodingString6 = new String ();
    String jFloodingString7 = new String ();String jFloodingString8 = new
String ();String jFloodingString9 = new String ();
    String jFloodingString10 = new String ();String jFloodingString11 = new
String ();String jFloodingString12 = new String ();
    String jFloodingString13 = new String ();String jFloodingString14 = new
String ();

    String jSnorkelString1 = new String ();String jSnorkelString2 = new String
();String jSnorkelString3 = new String ();
    String jSnorkelString4 = new String ();String jSnorkelString5 = new String
();String jSnorkelString6 = new String ();
```



```

        String jSnorkelString7 = new String ();String jSnorkelString8 = new String
        ();String jSnorkelString9 = new String ();
        String jSnorkelString10 = new String ();String jSnorkelString11 = new
        String ();String jSnorkelString12 = new String ();
        String jSnorkelString13 = new String ();String jSnorkelString14 = new
        String ();

```

```

        String jVentilateString1 = new String ();String jVentilateString2 = new
        String ();String jVentilateString3 = new String ();
        String jVentilateString4 = new String ();String jVentilateString5 = new
        String ();String jVentilateString6 = new String ();
        String jVentilateString7 = new String ();String jVentilateString8 = new
        String ();String jVentilateString9 = new String ();
        String jVentilateString10 = new String ();String jVentilateString11 = new
        String ();String jVentilateString12 = new String ();
        String jVentilateString13 = new String ();String jVentilateString14 = new
        String ();

```

```

        String jToxicGasString1 = new String ();String jToxicGasString2 = new
        String ();String jToxicGasString3 = new String ();
        String jToxicGasString4 = new String ();String jToxicGasString5 = new
        String ();String jToxicGasString6 = new String ();
        String jToxicGasString7 = new String ();String jToxicGasString8 = new
        String ();String jToxicGasString9 = new String ();
        String jToxicGasString10 = new String ();String jToxicGasString11 = new
        String ();String jToxicGasString12 = new String ();
        String jToxicGasString13 = new String ();String jToxicGasString14 = new
        String ();

```

```

        String jREString1 = new String ();String jREString2 = new String ();String
        jREString3 = new String ();
        String jREString4 = new String ();String jREString5 = new String ();String
        jREString6 = new String ();
        String jREString7 = new String ();String jREString8 = new String ();String
        jREString9 = new String ();
        String jREString10 = new String ();String jREString11 = new String
        ();String jREString12 = new String ();
        String jREString13 = new String ();String jREString14 = new String ();

```

```

        String jDiveString1 = new String ();String jDiveString2 = new String
        ();String jDiveString3 = new String ();
        String jDiveString4 = new String ();String jDiveString5 = new String
        ();String jDiveString6 = new String ();
        String jDiveString7 = new String ();String jDiveString8 = new String
        ();String jDiveString9 = new String ();
        String jDiveString10 = new String ();String jDiveString11 = new String
        ();String jDiveString12 = new String ();
        String jDiveString13 = new String ();String jDiveString14 = new String ();

```

```

        public CompartmentRigsGui crGui;

```

```

        /**Fire Constructor
        *
        */
        public CompartmentRigs() {}

```

```

        public CompartmentRigs(boolean isServer) {
            crGui = new CompartmentRigsGui(isServer);

```

```

    }

    /** This constructor allows casting to the appropriate object depending on
the
    * its type. It should read in the same order that the toBytes places
    * onto the ByteArrayOutputStream.
    */

    public CompartmentRigs(byte aBuffer[])
    {

        ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
        DataInputStream dis = new DataInputStream(bis);

        try
        {
            typeOfObject = dis.readInt();
            version = dis.readInt();
            type = dis.readUTF();
            ownerName = dis.readUTF();
            multicastAddressOfOwner = dis.readUTF();
            machineName = dis.readUTF();
            timeSent = dis.readUTF();
            messageToSend = dis.readUTF();

            jMainString1 = dis.readUTF();jMainString2 = dis.readUTF();jMainString3
= dis.readUTF();
            jMainString4 = dis.readUTF();jMainString5 = dis.readUTF();jMainString6
= dis.readUTF();
            jMainString7 = dis.readUTF();

            jFireString1 = dis.readUTF();jFireString2 = dis.readUTF();jFireString3
= dis.readUTF();
            jFireString4 = dis.readUTF();jFireString5 = dis.readUTF();jFireString6
= dis.readUTF();
            jFireString7 = dis.readUTF();jFireString8 = dis.readUTF();jFireString9
= dis.readUTF();
            jFireString10 = dis.readUTF();jFireString11 =
dis.readUTF();jFireString12 = dis.readUTF();
            jFireString13 = dis.readUTF();jFireString14 = dis.readUTF();

            jFloodingString1 = dis.readUTF();jFloodingString2 =
dis.readUTF();jFloodingString3 = dis.readUTF();
            jFloodingString4 = dis.readUTF();jFloodingString5 =
dis.readUTF();jFloodingString6 = dis.readUTF();
            jFloodingString7 = dis.readUTF();jFloodingString8 =
dis.readUTF();jFloodingString9 = dis.readUTF();
            jFloodingString10 = dis.readUTF();jFloodingString11 =
dis.readUTF();jFloodingString12 = dis.readUTF();
            jFloodingString13 = dis.readUTF();jFloodingString14 = dis.readUTF();

            jSnorkelString1 = dis.readUTF();jSnorkelString2 =
dis.readUTF();jSnorkelString3 = dis.readUTF();
            jSnorkelString4 = dis.readUTF();jSnorkelString5 =
dis.readUTF();jSnorkelString6 = dis.readUTF();
            jSnorkelString7 = dis.readUTF();jSnorkelString8 =
dis.readUTF();jSnorkelString9 = dis.readUTF();
            jSnorkelString10 = dis.readUTF();jSnorkelString11 =
dis.readUTF();jSnorkelString12 = dis.readUTF();
            jSnorkelString13 = dis.readUTF();jSnorkelString14 = dis.readUTF();
        }
    }

```

```

        jVentilateString1 = dis.readUTF();jVentilateString2 =
dis.readUTF();jVentilateString3 = dis.readUTF();
        jVentilateString4 = dis.readUTF();jVentilateString5 =
dis.readUTF();jVentilateString6 = dis.readUTF();
        jVentilateString7 = dis.readUTF();jVentilateString8 =
dis.readUTF();jVentilateString9 = dis.readUTF();
        jVentilateString10 = dis.readUTF();jVentilateString11 =
dis.readUTF();jVentilateString12 = dis.readUTF();
        jVentilateString13 = dis.readUTF();jVentilateString14 = dis.readUTF();

        jToxicGasString1 = dis.readUTF();jToxicGasString2 =
dis.readUTF();jToxicGasString3 = dis.readUTF();
        jToxicGasString4 = dis.readUTF();jToxicGasString5 =
dis.readUTF();jToxicGasString6 = dis.readUTF();
        jToxicGasString7 = dis.readUTF();jToxicGasString8 =
dis.readUTF();jToxicGasString9 = dis.readUTF();
        jToxicGasString10 = dis.readUTF();jToxicGasString11 =
dis.readUTF();jToxicGasString12 = dis.readUTF();
        jToxicGasString13 = dis.readUTF();jToxicGasString14 = dis.readUTF();

        jREString1 = dis.readUTF();jREString2 = dis.readUTF();jREString3 =
dis.readUTF();
        jREString4 = dis.readUTF();jREString5 = dis.readUTF();jREString6 =
dis.readUTF();
        jREString7 = dis.readUTF();jREString8 = dis.readUTF();jREString9 =
dis.readUTF();
        jREString10 = dis.readUTF();jREString11 = dis.readUTF();jREString12 =
dis.readUTF();
        jREString13 = dis.readUTF();jREString14 = dis.readUTF();

        jDiveString1 = dis.readUTF();jDiveString2 = dis.readUTF();jDiveString3
= dis.readUTF();
        jDiveString4 = dis.readUTF();jDiveString5 = dis.readUTF();jDiveString6
= dis.readUTF();
        jDiveString7 = dis.readUTF();jDiveString8 = dis.readUTF();jDiveString9
= dis.readUTF();
        jDiveString10 = dis.readUTF();jDiveString11 =
dis.readUTF();jDiveString12 = dis.readUTF();
        jDiveString13 = dis.readUTF();jDiveString14 = dis.readUTF();

    }
    catch(Exception e)
    {
        System.out.println("Exception occurred in (byte[])");
    }
}

```

/** toBytes method converts all instance variables within the object to a
* stream to allow a byte buffer to be sent on the network
*/

```

public byte[] toBytes()
{
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(bos);

    try
    {
        dos.writeInt(typeOfObject);
        dos.writeInt(version);
        dos.writeUTF(type);
    }
}

```

```

        dos.writeUTF(ownerName);
        dos.writeUTF(multicastAddressOfOwner);
        dos.writeUTF(machineName);
        dos.writeUTF(timeSent);
        dos.writeUTF(messageToSend);

dos.writeUTF(jMainString1);dos.writeUTF(jMainString2);dos.writeUTF(jMainString3
);

dos.writeUTF(jMainString4);dos.writeUTF(jMainString5);dos.writeUTF(jMainString6
);
        dos.writeUTF(jMainString7);

dos.writeUTF(jFireString1);dos.writeUTF(jFireString2);dos.writeUTF(jFireString3
);

dos.writeUTF(jFireString4);dos.writeUTF(jFireString5);dos.writeUTF(jFireString6
);

dos.writeUTF(jFireString7);dos.writeUTF(jFireString8);dos.writeUTF(jFireString9
);

dos.writeUTF(jFireString10);dos.writeUTF(jFireString11);dos.writeUTF(jFireStrin
gl2);
        dos.writeUTF(jFireString13);dos.writeUTF(jFireString14);

dos.writeUTF(jFloodingString1);dos.writeUTF(jFloodingString2);dos.writeUTF(jFlo
odingString3);

dos.writeUTF(jFloodingString4);dos.writeUTF(jFloodingString5);dos.writeUTF(jFlo
odingString6);

dos.writeUTF(jFloodingString7);dos.writeUTF(jFloodingString8);dos.writeUTF(jFlo
odingString9);

dos.writeUTF(jFloodingString10);dos.writeUTF(jFloodingString11);dos.writeUTF(jF
loodingString12);
        dos.writeUTF(jFloodingString13);dos.writeUTF(jFloodingString14);

dos.writeUTF(jSnorkelString1);dos.writeUTF(jSnorkelString2);dos.writeUTF(jSnork
elString3);

dos.writeUTF(jSnorkelString4);dos.writeUTF(jSnorkelString5);dos.writeUTF(jSnork
elString6);

dos.writeUTF(jSnorkelString7);dos.writeUTF(jSnorkelString8);dos.writeUTF(jSnork
elString9);

dos.writeUTF(jSnorkelString10);dos.writeUTF(jSnorkelString11);dos.writeUTF(jSno
rkelString12);
        dos.writeUTF(jSnorkelString13);dos.writeUTF(jSnorkelString14);

dos.writeUTF(jVentilateString1);dos.writeUTF(jVentilateString2);dos.writeUTF(jV
entilateString3);

dos.writeUTF(jVentilateString4);dos.writeUTF(jVentilateString5);dos.writeUTF(jV
entilateString6);

dos.writeUTF(jVentilateString7);dos.writeUTF(jVentilateString8);dos.writeUTF(jV
entilateString9);

```

```

dos.writeUTF(jVentilateString10);dos.writeUTF(jVentilateString11);dos.writeUTF(
jVentilateString12);
    dos.writeUTF(jVentilateString13);dos.writeUTF(jVentilateString14);

dos.writeUTF(jToxicGasString1);dos.writeUTF(jToxicGasString2);dos.writeUTF(jTox
icGasString3);

dos.writeUTF(jToxicGasString4);dos.writeUTF(jToxicGasString5);dos.writeUTF(jTox
icGasString6);

dos.writeUTF(jToxicGasString7);dos.writeUTF(jToxicGasString8);dos.writeUTF(jTox
icGasString9);

dos.writeUTF(jToxicGasString10);dos.writeUTF(jToxicGasString11);dos.writeUTF(jT
oxicGasString12);
    dos.writeUTF(jToxicGasString13);dos.writeUTF(jToxicGasString14);

dos.writeUTF(jREString1);dos.writeUTF(jREString2);dos.writeUTF(jREString3);
dos.writeUTF(jREString4);dos.writeUTF(jREString5);dos.writeUTF(jREString6);
dos.writeUTF(jREString7);dos.writeUTF(jREString8);dos.writeUTF(jREString9);
dos.writeUTF(jREString10);dos.writeUTF(jREString11);dos.writeUTF(jREString12);
    dos.writeUTF(jREString13);dos.writeUTF(jREString14);

dos.writeUTF(jDiveString1);dos.writeUTF(jDiveString2);dos.writeUTF(jDiveString3
);
dos.writeUTF(jDiveString4);dos.writeUTF(jDiveString5);dos.writeUTF(jDiveString6
);
dos.writeUTF(jDiveString7);dos.writeUTF(jDiveString8);dos.writeUTF(jDiveString9
);
dos.writeUTF(jDiveString10);dos.writeUTF(jDiveString11);dos.writeUTF(jDiveStrin
g12);
    dos.writeUTF(jDiveString13);dos.writeUTF(jDiveString14);

    }
    catch(IOException ioe)
    {
        System.out.println("IOException - Unable to convert Fire Object to
Bytes");
        return null;
    }

    return bos.toByteArray();
}

public String[] getData(){
    String holdString[] = new String[6];

    holdString[0] = type;
    holdString[1] = ownerName;
    holdString[2] = multicastAddressOfOwner;
    holdString[3] = machineName;
    holdString[4] = timeSent;

```

```

        holdString[5] = messageToSend;
        return holdString;
    }

    public void copyStatus(CompartmentRigs cr){

        if(jMainString1.equalsIgnoreCase("true"))
        ){cr.crGui.jMainPanel.fire.setSelected(true);}
        else{cr.crGui.jMainPanel.fire.setSelected(false);}

        if(jMainString2.equalsIgnoreCase("true")){cr.crGui.jMainPanel.flooding.setSelected(true);}
        else{cr.crGui.jMainPanel.flooding.setSelected(false);}

        if(jMainString3.equalsIgnoreCase("true")){cr.crGui.jMainPanel.snorkel.setSelected(true);}
        else{cr.crGui.jMainPanel.snorkel.setSelected(false);}

        if(jMainString4.equalsIgnoreCase("true")){cr.crGui.jMainPanel.ventilate.setSelected(true);}
        else{cr.crGui.jMainPanel.ventilate.setSelected(false);}

        if(jMainString5.equalsIgnoreCase("true")){cr.crGui.jMainPanel.toxicGas.setSelected(true);}
        else{cr.crGui.jMainPanel.toxicGas.setSelected(false);}

        if(jMainString6.equalsIgnoreCase("true")){cr.crGui.jMainPanel.rfre.setSelected(true);}
        else{cr.crGui.jMainPanel.rfre.setSelected(false);}

        if(jMainString7.equalsIgnoreCase("true")){cr.crGui.jMainPanel.dive.setSelected(true);}
        else{cr.crGui.jMainPanel.dive.setSelected(false);}

        if(jFireString1.equalsIgnoreCase("true"))
        ){cr.crGui.jFirePanel.cses.setSelected(true);}
        else{cr.crGui.jFirePanel.cses.setSelected(false);}

        if(jFireString2.equalsIgnoreCase("true")){cr.crGui.jFirePanel.control.setSelected(true);}
        else{cr.crGui.jFirePanel.control.setSelected(false);}

        if(jFireString3.equalsIgnoreCase("true")){cr.crGui.jFirePanel.nav.setSelected(true);}
        else{cr.crGui.jFirePanel.nav.setSelected(false);}

        if(jFireString4.equalsIgnoreCase("true")){cr.crGui.jFirePanel.fcml.setSelected(true);}
        else{cr.crGui.jFirePanel.fcml.setSelected(false);}

        if(jFireString5.equalsIgnoreCase("true")){cr.crGui.jFirePanel.tr.setSelected(true);}
        else{cr.crGui.jFirePanel.tr.setSelected(false);}

        if(jFireString6.equalsIgnoreCase("true")){cr.crGui.jFirePanel.amr.setSelected(true);}
        else{cr.crGui.jFirePanel.amr.setSelected(false);}

        if(jFireString7.equalsIgnoreCase("true")){cr.crGui.jFirePanel.erul.setSelected(true);}
        else{cr.crGui.jFirePanel.erul.setSelected(false);}
    }

```

```

if(jFireString8.equalsIgnoreCase("true")){cr.crGui.jFirePanel.erml.setSelected(
true);}
else{cr.crGui.jFirePanel.erml.setSelected(false);}

if(jFireString9.equalsIgnoreCase("true")){cr.crGui.jFirePanel.erf.setSelected(t
rue);}
else{cr.crGui.jFirePanel.erf.setSelected(false);}

if(jFireString10.equalsIgnoreCase("true")){cr.crGui.jFirePanel.tglo.setSelected
(true);}
else{cr.crGui.jFirePanel.tglo.setSelected(false);}

if(jFireString11.equalsIgnoreCase("true")){cr.crGui.jFirePanel.cb.setSelected(t
rue);}
else{cr.crGui.jFirePanel.cb.setSelected(false);}

if(jFireString12.equalsIgnoreCase("true")){cr.crGui.jFirePanel.msw.setSelected(
true);}
else{cr.crGui.jFirePanel.msw.setSelected(false);}

if(jFireString13.equalsIgnoreCase("true")){cr.crGui.jFirePanel.sa.setSelected(t
rue);}
else{cr.crGui.jFirePanel.sa.setSelected(false);}

if(jFireString14.equalsIgnoreCase("true")){cr.crGui.jFirePanel.totalRig.setSele
cted(true);}
else{cr.crGui.jFirePanel.totalRig.setSelected(false);}

if(jFloodingString1.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.cses.setS
electd(true);}
else{cr.crGui.jFloodingPanel.cses.setSelected(false);}

if(jFloodingString2.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.control.s
etSelected(true);}
else{cr.crGui.jFloodingPanel.control.setSelected(false);}

if(jFloodingString3.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.nav.setSe
lected(true);}
else{cr.crGui.jFloodingPanel.nav.setSelected(false);}

if(jFloodingString4.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.fcml.setS
electd(true);}
else{cr.crGui.jFloodingPanel.fcml.setSelected(false);}

if(jFloodingString5.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.tr.setSel
ected(true);}
else{cr.crGui.jFloodingPanel.tr.setSelected(false);}

if(jFloodingString6.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.amr.setSe
lected(true);}
else{cr.crGui.jFloodingPanel.amr.setSelected(false);}

if(jFloodingString7.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.erul.setS
electd(true);}
else{cr.crGui.jFloodingPanel.erul.setSelected(false);}

if(jFloodingString8.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.erml.setS
electd(true);}
else{cr.crGui.jFloodingPanel.erml.setSelected(false);}

if(jFloodingString9.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.erf.setSe
lected(true);}
else{cr.crGui.jFloodingPanel.erf.setSelected(false);}

```

```

if(jFloodingString10.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.tglo.setSelected(true);}
else{cr.crGui.jFloodingPanel.tglo.setSelected(false);}

if(jFloodingString11.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.cb.setSelected(true);}
else{cr.crGui.jFloodingPanel.cb.setSelected(false);}

if(jFloodingString12.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.msw.setSelected(true);}
else{cr.crGui.jFloodingPanel.msw.setSelected(false);}

if(jFloodingString13.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.sa.setSelected(true);}
else{cr.crGui.jFloodingPanel.sa.setSelected(false);}

if(jFloodingString14.equalsIgnoreCase("true")){cr.crGui.jFloodingPanel.totalRig.setSelected(true);}
else{cr.crGui.jFloodingPanel.totalRig.setSelected(false);}

if(jSnorkelString1.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.cses.setSelected(true);}
else{cr.crGui.jSnorkelPanel.cses.setSelected(false);}

if(jSnorkelString2.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.control.setSelected(true);}
else{cr.crGui.jSnorkelPanel.control.setSelected(false);}

if(jSnorkelString3.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.nav.setSelected(true);}
else{cr.crGui.jSnorkelPanel.nav.setSelected(false);}

if(jSnorkelString4.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.fcml.setSelected(true);}
else{cr.crGui.jSnorkelPanel.fcml.setSelected(false);}

if(jSnorkelString5.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.tr.setSelected(true);}
else{cr.crGui.jSnorkelPanel.tr.setSelected(false);}

if(jSnorkelString6.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.amr.setSelected(true);}
else{cr.crGui.jSnorkelPanel.amr.setSelected(false);}

if(jSnorkelString7.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.erul.setSelected(true);}
else{cr.crGui.jSnorkelPanel.erul.setSelected(false);}

if(jSnorkelString8.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.erml.setSelected(true);}
else{cr.crGui.jSnorkelPanel.erml.setSelected(false);}

if(jSnorkelString9.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.erf.setSelected(true);}
else{cr.crGui.jSnorkelPanel.erf.setSelected(false);}

if(jSnorkelString10.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.tglo.setSelected(true);}
else{cr.crGui.jSnorkelPanel.tglo.setSelected(false);}

if(jSnorkelString11.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.cb.setSelected(true);}
else{cr.crGui.jSnorkelPanel.cb.setSelected(false);}

```



```

if(jSnorkelString12.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.msw.setSelected(true);}
else{cr.crGui.jSnorkelPanel.msw.setSelected(false);}

if(jSnorkelString13.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.sa.setSelected(true);}
else{cr.crGui.jSnorkelPanel.sa.setSelected(false);}

if(jSnorkelString14.equalsIgnoreCase("true")){cr.crGui.jSnorkelPanel.totalRig.setSelected(true);}
else{cr.crGui.jSnorkelPanel.totalRig.setSelected(false);}

if(jVentilateString1.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.cses.setSelected(true);}
else{cr.crGui.jVentilatePanel.cses.setSelected(false);}

if(jVentilateString2.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.control.setSelected(true);}
else{cr.crGui.jVentilatePanel.control.setSelected(false);}

if(jVentilateString3.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.nav.setSelected(true);}
else{cr.crGui.jVentilatePanel.nav.setSelected(false);}

if(jVentilateString4.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.fcml.setSelected(true);}
else{cr.crGui.jVentilatePanel.fcml.setSelected(false);}

if(jVentilateString5.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.tr.setSelected(true);}
else{cr.crGui.jVentilatePanel.tr.setSelected(false);}

if(jVentilateString6.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.amr.setSelected(true);}
else{cr.crGui.jVentilatePanel.amr.setSelected(false);}

if(jVentilateString7.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.erul.setSelected(true);}
else{cr.crGui.jVentilatePanel.erul.setSelected(false);}

if(jVentilateString8.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.erml.setSelected(true);}
else{cr.crGui.jVentilatePanel.erml.setSelected(false);}

if(jVentilateString9.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.erf.setSelected(true);}
else{cr.crGui.jVentilatePanel.erf.setSelected(false);}

if(jVentilateString10.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.tglo.setSelected(true);}
else{cr.crGui.jVentilatePanel.tglo.setSelected(false);}

if(jVentilateString11.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.cb.setSelected(true);}
else{cr.crGui.jVentilatePanel.cb.setSelected(false);}

if(jVentilateString12.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.msw.setSelected(true);}
else{cr.crGui.jVentilatePanel.msw.setSelected(false);}

if(jVentilateString13.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.sa.setSelected(true);}
else{cr.crGui.jVentilatePanel.sa.setSelected(false);}

```

```

if(jVentilateString14.equalsIgnoreCase("true")){cr.crGui.jVentilatePanel.totalRig.setSelected(true);}
else{cr.crGui.jVentilatePanel.totalRig.setSelected(false);}

if(jToxicGasString1.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.cses.setSelected(true);}
else{cr.crGui.jToxicGasPanel.cses.setSelected(false);}

if(jToxicGasString2.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.control.setSelected(true);}
else{cr.crGui.jToxicGasPanel.control.setSelected(false);}

if(jToxicGasString3.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.nav.setSelected(true);}
else{cr.crGui.jToxicGasPanel.nav.setSelected(false);}

if(jToxicGasString4.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.fcml.setSelected(true);}
else{cr.crGui.jToxicGasPanel.fcml.setSelected(false);}

if(jToxicGasString5.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.tr.setSelected(true);}
else{cr.crGui.jToxicGasPanel.tr.setSelected(false);}

if(jToxicGasString6.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.amr.setSelected(true);}
else{cr.crGui.jToxicGasPanel.amr.setSelected(false);}

if(jToxicGasString7.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.erul.setSelected(true);}
else{cr.crGui.jToxicGasPanel.erul.setSelected(false);}

if(jToxicGasString8.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.erml.setSelected(true);}
else{cr.crGui.jToxicGasPanel.erml.setSelected(false);}

if(jToxicGasString9.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.erf.setSelected(true);}
else{cr.crGui.jToxicGasPanel.erf.setSelected(false);}

if(jToxicGasString10.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.tglo.setSelected(true);}
else{cr.crGui.jToxicGasPanel.tglo.setSelected(false);}

if(jToxicGasString11.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.cb.setSelected(true);}
else{cr.crGui.jToxicGasPanel.cb.setSelected(false);}

if(jToxicGasString12.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.msw.setSelected(true);}
else{cr.crGui.jToxicGasPanel.msw.setSelected(false);}

if(jToxicGasString13.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.sa.setSelected(true);}
else{cr.crGui.jToxicGasPanel.sa.setSelected(false);}

if(jToxicGasString14.equalsIgnoreCase("true")){cr.crGui.jToxicGasPanel.totalRig.setSelected(true);}
else{cr.crGui.jToxicGasPanel.totalRig.setSelected(false);}

if(jREString1.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.cses.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.cses.setSelected(false);}

```

```

if(jREString2.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.control.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.control.setSelected(false);}

if(jREString3.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.nav.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.nav.setSelected(false);}

if(jREString4.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.fcml.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.fcml.setSelected(false);}

if(jREString5.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.tr.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.tr.setSelected(false);}

if(jREString6.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.amr.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.amr.setSelected(false);}

if(jREString7.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.erul.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.erul.setSelected(false);}

if(jREString8.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.erml.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.erml.setSelected(false);}

if(jREString9.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.erf.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.erf.setSelected(false);}

if(jREString10.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.tglo.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.tglo.setSelected(false);}

if(jREString11.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.cb.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.cb.setSelected(false);}

if(jREString12.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.msw.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.msw.setSelected(false);}

if(jREString13.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.sa.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.sa.setSelected(false);}

if(jREString14.equalsIgnoreCase("true")){cr.crGui.jReducedElectricalPanel.totalRig.setSelected(true);}
else{cr.crGui.jReducedElectricalPanel.totalRig.setSelected(false);}

if(jDiveString1.equalsIgnoreCase("true")){cr.crGui.jDivePanel.cses.setSelected(true);}
else{cr.crGui.jDivePanel.cses.setSelected(false);}

if(jDiveString2.equalsIgnoreCase("true")){cr.crGui.jDivePanel.control.setSelected(true);}
else{cr.crGui.jDivePanel.control.setSelected(false);}

if(jDiveString3.equalsIgnoreCase("true")){cr.crGui.jDivePanel.nav.setSelected(true);}
else{cr.crGui.jDivePanel.nav.setSelected(false);}

```

```

if(jDiveString4.equalsIgnoreCase("true")){cr.crGui.jDivePanel.fcml.setSelected(
true);}
    else{cr.crGui.jDivePanel.fcml.setSelected(false);}

if(jDiveString5.equalsIgnoreCase("true")){cr.crGui.jDivePanel.tr.setSelected(tr
ue);}
    else{cr.crGui.jDivePanel.tr.setSelected(false);}

if(jDiveString6.equalsIgnoreCase("true")){cr.crGui.jDivePanel.amr.setSelected(t
rue);}
    else{cr.crGui.jDivePanel.amr.setSelected(false);}

if(jDiveString7.equalsIgnoreCase("true")){cr.crGui.jDivePanel.erul.setSelected(
true);}
    else{cr.crGui.jDivePanel.erul.setSelected(false);}

if(jDiveString8.equalsIgnoreCase("true")){cr.crGui.jDivePanel.erml.setSelected(
true);}
    else{cr.crGui.jDivePanel.erml.setSelected(false);}

if(jDiveString9.equalsIgnoreCase("true")){cr.crGui.jDivePanel.erf.setSelected(t
rue);}
    else{cr.crGui.jDivePanel.erf.setSelected(false);}

if(jDiveString10.equalsIgnoreCase("true")){cr.crGui.jDivePanel.tglo.setSelected
(true);}
    else{cr.crGui.jDivePanel.tglo.setSelected(false);}

if(jDiveString11.equalsIgnoreCase("true")){cr.crGui.jDivePanel.cb.setSelected(t
rue);}
    else{cr.crGui.jDivePanel.cb.setSelected(false);}

if(jDiveString12.equalsIgnoreCase("true")){cr.crGui.jDivePanel.msw.setSelected(
true);}
    else{cr.crGui.jDivePanel.msw.setSelected(false);}

if(jDiveString13.equalsIgnoreCase("true")){cr.crGui.jDivePanel.sa.setSelected(t
rue);}
    else{cr.crGui.jDivePanel.sa.setSelected(false);}

if(jDiveString14.equalsIgnoreCase("true")){cr.crGui.jDivePanel.totalRig.setSele
cted(true);}
    else{cr.crGui.jDivePanel.totalRig.setSelected(false);}

}

public void setOwner (String A, String B){
    ownerName = A;
    multicastAddressOfOwner = B;
}

public void setData (String A, String B, String C){
    machineName = A;
    timeSent = B;
    messageToSend = C;
}

```

```

    }

    public void setStatusFromGui(){
        if(crGui.jMainPanel.fire.isSelected() == true){jMainString1 = "true";}
        else{jMainString1 = "false";}
        if(crGui.jMainPanel.flooding.isSelected() == true){jMainString2 =
"true";}
        else{jMainString2 = "false";}
        if(crGui.jMainPanel.snorkel.isSelected() == true){jMainString3 =
"true";}
        else{jMainString3 = "false";}
        if(crGui.jMainPanel.ventilate.isSelected() == true){jMainString4 =
"true";}
        else{jMainString4 = "false";}
        if(crGui.jMainPanel.toxicGas.isSelected() == true){jMainString5 =
"true";}
        else{jMainString5 = "false";}
        if(crGui.jMainPanel.rfire.isSelected() == true){jMainString6 = "true";}
        else{jMainString6 = "false";}
        if(crGui.jMainPanel.dive.isSelected() == true){jMainString7 = "true";}
        else{jMainString7 = "false";}

        if(crGui.jFirePanel.cses.isSelected() == true){jFireString1 = "true";}
        else{jFireString1 = "false";}
        if(crGui.jFirePanel.control.isSelected() == true){jFireString2 =
"true";}
        else{jFireString2 = "false";}
        if(crGui.jFirePanel.nav.isSelected() == true){jFireString3 = "true";}
        else{jFireString3 = "false";}
        if(crGui.jFirePanel.fcml.isSelected() == true){jFireString4 = "true";}
        else{jFireString4 = "false";}
        if(crGui.jFirePanel.tr.isSelected() == true){jFireString5 = "true";}
        else{jFireString5 = "false";}
        if(crGui.jFirePanel.amr.isSelected() == true){jFireString6 = "true";}
        else{jFireString6 = "false";}
        if(crGui.jFirePanel.erul.isSelected() == true){jFireString7 = "true";}
        else{jFireString7 = "false";}
        if(crGui.jFirePanel.erml.isSelected() == true){jFireString8 = "true";}
        else{jFireString8 = "false";}
        if(crGui.jFirePanel.erf.isSelected() == true){jFireString9 = "true";}
        else{jFireString9 = "false";}
        if(crGui.jFirePanel.tglo.isSelected() == true){jFireString10 = "true";}
        else{jFireString10 = "false";}
        if(crGui.jFirePanel.cb.isSelected() == true){jFireString11 = "true";}
        else{jFireString11 = "false";}
        if(crGui.jFirePanel.msw.isSelected() == true){jFireString12 = "true";}
        else{jFireString12 = "false";}
        if(crGui.jFirePanel.sa.isSelected() == true){jFireString13 = "true";}
        else{jFireString13 = "false";}
        if(crGui.jFirePanel.totalRig.isSelected() == true){jFireString14 =
"true";}
        else{jFireString14 = "false";}

        if(crGui.jFloodingPanel.cses.isSelected() == true){jFloodingString1 =
"true";}
        else{jFloodingString1 = "false";}
        if(crGui.jFloodingPanel.control.isSelected() == true){jFloodingString2
= "true";}
        else{jFloodingString2 = "false";}
        if(crGui.jFloodingPanel.nav.isSelected() == true){jFloodingString3 =
"true";}
        else{jFloodingString3 = "false";}

```

```

        if(crGui.jFloodingPanel.fcml.isSelected() == true){jFloodingString4 =
"true";}
        else{jFloodingString4 = "false";}
        if(crGui.jFloodingPanel.tr.isSelected() == true){jFloodingString5 =
"true";}
        else{jFloodingString5 = "false";}
        if(crGui.jFloodingPanel.amr.isSelected() == true){jFloodingString6 =
"true";}
        else{jFloodingString6 = "false";}
        if(crGui.jFloodingPanel.erul.isSelected() == true){jFloodingString7 =
"true";}
        else{jFloodingString7 = "false";}
        if(crGui.jFloodingPanel.erml.isSelected() == true){jFloodingString8 =
"true";}
        else{jFloodingString8 = "false";}
        if(crGui.jFloodingPanel.erf.isSelected() == true){jFloodingString9 =
"true";}
        else{jFloodingString9 = "false";}
        if(crGui.jFloodingPanel.tglo.isSelected() == true){jFloodingString10 =
"true";}
        else{jFloodingString10 = "false";}
        if(crGui.jFloodingPanel.cb.isSelected() == true){jFloodingString11 =
"true";}
        else{jFloodingString11 = "false";}
        if(crGui.jFloodingPanel.msw.isSelected() == true){jFloodingString12 =
"true";}
        else{jFloodingString12 = "false";}
        if(crGui.jFloodingPanel.sa.isSelected() == true){jFloodingString13 =
"true";}
        else{jFloodingString13 = "false";}
        if(crGui.jFloodingPanel.totalRig.isSelected() ==
true){jFloodingString14 = "true";}
        else{jFloodingString14 = "false";}

        if(crGui.jSnorkelPanel.cses.isSelected() == true){jSnorkelString1 =
"true";}
        else{jSnorkelString1 = "false";}
        if(crGui.jSnorkelPanel.control.isSelected() == true){jSnorkelString2 =
"true";}
        else{jSnorkelString2 = "false";}
        if(crGui.jSnorkelPanel.nav.isSelected() == true){jSnorkelString3 =
"true";}
        else{jSnorkelString3 = "false";}
        if(crGui.jSnorkelPanel.fcml.isSelected() == true){jSnorkelString4 =
"true";}
        else{jSnorkelString4 = "false";}
        if(crGui.jSnorkelPanel.tr.isSelected() == true){jSnorkelString5 =
"true";}
        else{jSnorkelString5 = "false";}
        if(crGui.jSnorkelPanel.amr.isSelected() == true){jSnorkelString6 =
"true";}
        else{jSnorkelString6 = "false";}
        if(crGui.jSnorkelPanel.erul.isSelected() == true){jSnorkelString7 =
"true";}
        else{jSnorkelString7 = "false";}
        if(crGui.jSnorkelPanel.erml.isSelected() == true){jSnorkelString8 =
"true";}
        else{jSnorkelString8 = "false";}
        if(crGui.jSnorkelPanel.erf.isSelected() == true){jSnorkelString9 =
"true";}
        else{jSnorkelString9 = "false";}
        if(crGui.jSnorkelPanel.tglo.isSelected() == true){jSnorkelString10 =
"true";}
        else{jSnorkelString10 = "false";}
        if(crGui.jSnorkelPanel.cb.isSelected() == true){jSnorkelString11 =
"true";}
        else{jSnorkelString11 = "false";}

```

```

        if(crGui.jSnorkelPanel.msw.isSelected() == true){jSnorkelString12 =
"true";}
        else{jSnorkelString12 = "false";}
        if(crGui.jSnorkelPanel.sa.isSelected() == true){jSnorkelString13 =
"true";}
        else{jSnorkelString13 = "false";}
        if(crGui.jSnorkelPanel.totalRig.isSelected() == true){jSnorkelString14
= "true";}
        else{jSnorkelString14 = "false";}

        if(crGui.jVentilatePanel.cses.isSelected() == true){jVentilateString1 =
"true";}
        else{jVentilateString1 = "false";}
        if(crGui.jVentilatePanel.control.isSelected() ==
true){jVentilateString2 = "true";}
        else{jVentilateString2 = "false";}
        if(crGui.jVentilatePanel.nav.isSelected() == true){jVentilateString3 =
"true";}
        else{jVentilateString3 = "false";}
        if(crGui.jVentilatePanel.fcml.isSelected() == true){jVentilateString4 =
"true";}
        else{jVentilateString4 = "false";}
        if(crGui.jVentilatePanel.tr.isSelected() == true){jVentilateString5 =
"true";}
        else{jVentilateString5 = "false";}
        if(crGui.jVentilatePanel.amr.isSelected() == true){jVentilateString6 =
"true";}
        else{jVentilateString6 = "false";}
        if(crGui.jVentilatePanel.erul.isSelected() == true){jVentilateString7 =
"true";}
        else{jVentilateString7 = "false";}
        if(crGui.jVentilatePanel.erml.isSelected() == true){jVentilateString8 =
"true";}
        else{jVentilateString8 = "false";}
        if(crGui.jVentilatePanel.erf.isSelected() == true){jVentilateString9 =
"true";}
        else{jVentilateString9 = "false";}
        if(crGui.jVentilatePanel.tglo.isSelected() == true){jVentilateString10
= "true";}
        else{jVentilateString10 = "false";}
        if(crGui.jVentilatePanel.cb.isSelected() == true){jVentilateString11 =
"true";}
        else{jVentilateString11 = "false";}
        if(crGui.jVentilatePanel.msw.isSelected() == true){jVentilateString12 =
"true";}
        else{jVentilateString12 = "false";}
        if(crGui.jVentilatePanel.sa.isSelected() == true){jVentilateString13 =
"true";}
        else{jVentilateString13 = "false";}
        if(crGui.jVentilatePanel.totalRig.isSelected() ==
true){jVentilateString14 = "true";}
        else{jVentilateString14 = "false";}

        if(crGui.jToxicGasPanel.cses.isSelected() == true){jToxicGasString1 =
"true";}
        else{jToxicGasString1 = "false";}
        if(crGui.jToxicGasPanel.control.isSelected() == true){jToxicGasString2
= "true";}
        else{jToxicGasString2 = "false";}
        if(crGui.jToxicGasPanel.nav.isSelected() == true){jToxicGasString3 =
"true";}
        else{jToxicGasString3 = "false";}
        if(crGui.jToxicGasPanel.fcml.isSelected() == true){jToxicGasString4 =
"true";}
        else{jToxicGasString4 = "false";}
        if(crGui.jToxicGasPanel.tr.isSelected() == true){jToxicGasString5 =
"true";}

```

```

        else{jToxicGasString5 = "false";}
        if(crGui.jToxicGasPanel.amr.isSelected() == true){jToxicGasString6 =
"true";}
        else{jToxicGasString6 = "false";}
        if(crGui.jToxicGasPanel.erul.isSelected() == true){jToxicGasString7 =
"true";}
        else{jToxicGasString7 = "false";}
        if(crGui.jToxicGasPanel.erml.isSelected() == true){jToxicGasString8 =
"true";}
        else{jToxicGasString8 = "false";}
        if(crGui.jToxicGasPanel.erf.isSelected() == true){jToxicGasString9 =
"true";}
        else{jToxicGasString9 = "false";}
        if(crGui.jToxicGasPanel.tglo.isSelected() == true){jToxicGasString10 =
"true";}
        else{jToxicGasString10 = "false";}
        if(crGui.jToxicGasPanel.cb.isSelected() == true){jToxicGasString11 =
"true";}
        else{jToxicGasString11 = "false";}
        if(crGui.jToxicGasPanel.msw.isSelected() == true){jToxicGasString12 =
"true";}
        else{jToxicGasString12 = "false";}
        if(crGui.jToxicGasPanel.sa.isSelected() == true){jToxicGasString13 =
"true";}
        else{jToxicGasString13 = "false";}
        if(crGui.jToxicGasPanel.totalRig.isSelected() ==
true){jToxicGasString14 = "true";}
        else{jToxicGasString14 = "false";}

        if(crGui.jReducedElectricalPanel.cses.isSelected() == true){jREString1
= "true";}
        else{jREString1 = "false";}
        if(crGui.jReducedElectricalPanel.control.isSelected() ==
true){jREString2 = "true";}
        else{jREString2 = "false";}
        if(crGui.jReducedElectricalPanel.nav.isSelected() == true){jREString3 =
"true";}
        else{jREString3 = "false";}
        if(crGui.jReducedElectricalPanel.fcml.isSelected() == true){jREString4
= "true";}
        else{jREString4 = "false";}
        if(crGui.jReducedElectricalPanel.tr.isSelected() == true){jREString5 =
"true";}
        else{jREString5 = "false";}
        if(crGui.jReducedElectricalPanel.amr.isSelected() == true){jREString6 =
"true";}
        else{jREString6 = "false";}
        if(crGui.jReducedElectricalPanel.erul.isSelected() == true){jREString7
= "true";}
        else{jREString7 = "false";}
        if(crGui.jReducedElectricalPanel.erml.isSelected() == true){jREString8
= "true";}
        else{jREString8 = "false";}
        if(crGui.jReducedElectricalPanel.erf.isSelected() == true){jREString9 =
"true";}
        else{jREString9 = "false";}
        if(crGui.jReducedElectricalPanel.tglo.isSelected() == true){jREString10
= "true";}
        else{jREString10 = "false";}
        if(crGui.jReducedElectricalPanel.cb.isSelected() == true){jREString11 =
"true";}
        else{jREString11 = "false";}
        if(crGui.jReducedElectricalPanel.msw.isSelected() == true){jREString12
= "true";}
        else{jREString12 = "false";}
        if(crGui.jReducedElectricalPanel.sa.isSelected() == true){jREString13 =
"true";}

```



```

        else{jREString13 = "false";}
        if(crGui.jReducedElectricalPanel.totalRig.isSelected() ==
true){jREString14 = "true";}
        else{jREString14 = "false";}

        if(crGui.jDivePanel.cses.isSelected() == true){jDiveString1 = "true";}
        else{jDiveString1 = "false";}
        if(crGui.jDivePanel.control.isSelected() == true){jDiveString2 =
"true";}
        else{jDiveString2 = "false";}
        if(crGui.jDivePanel.nav.isSelected() == true){jDiveString3 = "true";}
        else{jDiveString3 = "false";}
        if(crGui.jDivePanel.fcml.isSelected() == true){jDiveString4 = "true";}
        else{jDiveString4 = "false";}
        if(crGui.jDivePanel.tr.isSelected() == true){jDiveString5 = "true";}
        else{jDiveString5 = "false";}
        if(crGui.jDivePanel.amr.isSelected() == true){jDiveString6 = "true";}
        else{jDiveString6 = "false";}
        if(crGui.jDivePanel.erul.isSelected() == true){jDiveString7 = "true";}
        else{jDiveString7 = "false";}
        if(crGui.jDivePanel.erml.isSelected() == true){jDiveString8 = "true";}
        else{jDiveString8 = "false";}
        if(crGui.jDivePanel.erf.isSelected() == true){jDiveString9 = "true";}
        else{jDiveString9 = "false";}
        if(crGui.jDivePanel.tglo.isSelected() == true){jDiveString10 = "true";}
        else{jDiveString10 = "false";}
        if(crGui.jDivePanel.cb.isSelected() == true){jDiveString11 = "true";}
        else{jDiveString11 = "false";}
        if(crGui.jDivePanel.msw.isSelected() == true){jDiveString12 = "true";}
        else{jDiveString12 = "false";}
        if(crGui.jDivePanel.sa.isSelected() == true){jDiveString13 = "true";}
        else{jDiveString13 = "false";}
        if(crGui.jDivePanel.totalRig.isSelected() == true){jDiveString14 =
"true";}
        else{jDiveString14 = "false";}

    }

}

```

Class: ShipAtmospheres

```
package swipNet.shipObjects;

import java.io.*;
import java.util.*;
import swipNet.gui.*;

/**
 * A ship object that holds information within strings that can
 * be directly sent over the network for ShipAtmospheres parameters<p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class ShipAtmospheres extends Object implements Serializable{

    static int typeOfObject = 23;
    static int version = 1;
    static String type = new String ("SHIPATMOSPHERES");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");//Fix With
Function Call
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    String mainString1 = new String();String mainString2 = new String();String
mainString3 = new String();
    String mainString4 = new String();String mainString5 = new String();String
mainString6 = new String();

    String fcString1 = new String();String fcString2 = new String();String
fcString3 = new String();
    String fcString4 = new String();String fcString5 = new String();String
fcString6 = new String();
    String fcString7 = new String();String fcString8 = new String();String
fcString9 = new String();

    String erString1 = new String();String erString2 = new String();String
erString3 = new String();
    String erString4 = new String();String erString5 = new String();String
erString6 = new String();
    String erString7 = new String();String erString8 = new String();String
erString9 = new String();

    public ShipAtmospheresGui saGui;

    /**Fire Constructor
     *
     */
    public ShipAtmospheres() {}

    public ShipAtmospheres(boolean isServer) {
        saGui = new ShipAtmospheresGui(isServer);
    }
}
```

```

the /** This constructor allows casting to the appropriate object depending on
    * its type. It should read in the same order that the toBytes places
    * onto the ByteArrayOutputStream.
    */

public ShipAtmospheres(byte aBuffer[])
{
    ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
    DataInputStream dis = new DataInputStream(bis);

    try
    {
        typeOfObject = dis.readInt();
        version = dis.readInt();
        type = dis.readUTF();
        ownerName = dis.readUTF();
        multicastAddressOfOwner = dis.readUTF();
        machineName = dis.readUTF();
        timeSent = dis.readUTF();
        messageToSend = dis.readUTF();

        mainString1 = dis.readUTF();mainString2 = dis.readUTF();mainString3 =
dis.readUTF();
        mainString4 = dis.readUTF();mainString5 = dis.readUTF();mainString6 =
dis.readUTF();

        fcString1 = dis.readUTF();fcString2 = dis.readUTF();fcString3 =
dis.readUTF();
        fcString4 = dis.readUTF();fcString5 = dis.readUTF();fcString6 =
dis.readUTF();
        fcString7 = dis.readUTF();fcString8 = dis.readUTF();fcString9 =
dis.readUTF();

        erString1 = dis.readUTF();erString2 = dis.readUTF();erString3 =
dis.readUTF();
        erString4 = dis.readUTF();erString5 = dis.readUTF();erString6 =
dis.readUTF();
        erString7 = dis.readUTF();erString8 = dis.readUTF();erString9 =
dis.readUTF();

    }
    catch(Exception e)
    {
        System.out.println("Exception occurred in (byte[])");
    }
}

/** toBytes method converts all instance variables within the object to a
 * stream to allow a byte buffer to be sent on the network
 */

public byte[] toBytes()
{
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(bos);

    try

```

```

        {
            dos.writeInt(typeOfObject);
            dos.writeInt(version);
            dos.writeUTF(type);

            dos.writeUTF(ownerName);
            dos.writeUTF(multicastAddressOfOwner);
            dos.writeUTF(machineName);
            dos.writeUTF(timeSent);
            dos.writeUTF(messageToSend);

dos.writeUTF(mainString1);dos.writeUTF(mainString2);dos.writeUTF(mainString3);
dos.writeUTF(mainString4);dos.writeUTF(mainString5);dos.writeUTF(mainString6);

dos.writeUTF(fcString1);dos.writeUTF(fcString2);dos.writeUTF(fcString3);
dos.writeUTF(fcString4);dos.writeUTF(fcString5);dos.writeUTF(fcString6);
dos.writeUTF(fcString7);dos.writeUTF(fcString8);dos.writeUTF(fcString9);

dos.writeUTF(erString1);dos.writeUTF(erString2);dos.writeUTF(erString3);
dos.writeUTF(erString4);dos.writeUTF(erString5);dos.writeUTF(erString6);
dos.writeUTF(erString7);dos.writeUTF(erString8);dos.writeUTF(erString9);
        }
        catch(IOException ioe)
        {
            System.out.println("IOException - Unable to convert Fire Object to
Bytes");
            return null;
        }

        return bos.toByteArray();
    }

    public String[] getData(){
        String holdString[] = new String[6];

        holdString[0] = type;
        holdString[1] = ownerName;
        holdString[2] = multicastAddressOfOwner;
        holdString[3] = machineName;
        holdString[4] = timeSent;
        holdString[5] = messageToSend;

        return holdString;
    }

    public void copyStatus(ShipAtmospheres sa){

if(mainString1.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limitOnehFCCB.setS
elected(true);}
        else{sa.saGui.jMainPanel.limitOnehFCCB.setSelected(false);}
    }

```

```

if(mainString2.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limit24hFCCB.setSelected(true);}
else{sa.saGui.jMainPanel.limit24hFCCB.setSelected(false);}

if(mainString3.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limit90dFCCB.setSelected(true);}
else{sa.saGui.jMainPanel.limit90dFCCB.setSelected(false);}

if(mainString4.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limitOnehERCB.setSelected(true);}
else{sa.saGui.jMainPanel.limitOnehERCB.setSelected(false);}

if(mainString5.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limit24hERCB.setSelected(true);}
else{sa.saGui.jMainPanel.limit24hERCB.setSelected(false);}

if(mainString6.equalsIgnoreCase("true")){sa.saGui.jMainPanel.limit90dERCB.setSelected(true);}
else{sa.saGui.jMainPanel.limit90dERCB.setSelected(false);}

sa.saGui.jFCPanel.o2TF.setText(fcString1);
sa.saGui.jFCPanel.co2TF.setText(fcString2);
sa.saGui.jFCPanel.r114TF.setText(fcString3);
sa.saGui.jFCPanel.h2TF.setText(fcString4);
sa.saGui.jFCPanel.coTF.setText(fcString5);
sa.saGui.jFCPanel.ottoTF.setText(fcString6);

if(fcString7.equalsIgnoreCase("true")){sa.saGui.jFCPanel.limitOnehCB.setSelected(true);}
else{sa.saGui.jFCPanel.limitOnehCB.setSelected(false);}

if(fcString8.equalsIgnoreCase("true")){sa.saGui.jFCPanel.limit24hCB.setSelected(true);}
else{sa.saGui.jFCPanel.limit24hCB.setSelected(false);}

if(fcString9.equalsIgnoreCase("true")){sa.saGui.jFCPanel.limit90dCB.setSelected(true);}
else{sa.saGui.jFCPanel.limit90dCB.setSelected(false);}

sa.saGui.jERPanel.o2TF.setText(erString1);
sa.saGui.jERPanel.co2TF.setText(erString2);
sa.saGui.jERPanel.r114TF.setText(erString3);
sa.saGui.jERPanel.h2TF.setText(erString4);
sa.saGui.jERPanel.coTF.setText(erString5);
sa.saGui.jERPanel.ottoTF.setText(erString6);

if(erString7.equalsIgnoreCase("true")){sa.saGui.jERPanel.limitOnehCB.setSelected(true);}
else{sa.saGui.jERPanel.limitOnehCB.setSelected(false);}

if(erString8.equalsIgnoreCase("true")){sa.saGui.jERPanel.limit24hCB.setSelected(true);}
else{sa.saGui.jERPanel.limit24hCB.setSelected(false);}

if(erString9.equalsIgnoreCase("true")){sa.saGui.jERPanel.limit90dCB.setSelected(true);}
else{sa.saGui.jERPanel.limit90dCB.setSelected(false);}

}

public void setOwner (String A, String B){
    ownerName = A;

```

```

        multicastAddressOfOwner = B;
    }

    public void setData (String A, String B, String C){
        machineName = A;
        timeSent = B;
        messageToSend = C;
    }

    public void setStatusFromGui(){
        if(saGui.jMainPanel.limitOnehFCCB.isSelected() == true){mainString1 =
"true";}
        else{mainString1 = "false";}
        if(saGui.jMainPanel.limit24hFCCB.isSelected() == true){mainString2 =
"true";}
        else{mainString2 = "false";}
        if(saGui.jMainPanel.limit90dFCCB.isSelected() == true){mainString3 =
"true";}
        else{mainString3 = "false";}
        if(saGui.jMainPanel.limitOnehERCB.isSelected() == true){mainString4 =
"true";}
        else{mainString4 = "false";}
        if(saGui.jMainPanel.limit24hERCB.isSelected() == true){mainString5 =
"true";}
        else{mainString5 = "false";}
        if(saGui.jMainPanel.limit90dERCB.isSelected() == true){mainString6 =
"true";}
        else{mainString6 = "false";}

        fcString1 = saGui.jFCPanel.o2TF.getText();
        fcString2 = saGui.jFCPanel.co2TF.getText();
        fcString3 = saGui.jFCPanel.r114TF.getText();
        fcString4 = saGui.jFCPanel.h2TF.getText();
        fcString5 = saGui.jFCPanel.coTF.getText();
        fcString6 = saGui.jFCPanel.ottoTF.getText();
        if(saGui.jFCPanel.limitOnehCB.isSelected() == true){fcString7 =
"true";}
        else{fcString7 = "false";}
        if(saGui.jFCPanel.limit24hCB.isSelected() == true){fcString8 =
"true";}
        else{fcString8 = "false";}
        if(saGui.jFCPanel.limit90dCB.isSelected() == true){fcString9 =
"true";}
        else{fcString9 = "false";}

        erString1 = saGui.jERPanel.o2TF.getText();
        erString2 = saGui.jERPanel.co2TF.getText();
        erString3 = saGui.jERPanel.r114TF.getText();
        erString4 = saGui.jERPanel.h2TF.getText();
        erString5 = saGui.jERPanel.coTF.getText();
        erString6 = saGui.jERPanel.ottoTF.getText();
        if(saGui.jERPanel.limitOnehCB.isSelected() == true){erString7 =
"true";}
        else{erString7 = "false";}
        if(saGui.jERPanel.limit24hCB.isSelected() == true){erString8 =
"true";}
        else{erString8 = "false";}
        if(saGui.jERPanel.limit90dCB.isSelected() == true){erString9 =
"true";}
        else{erString9 = "false";} }}

```

Package: swipNet.utility

Class: PostOffice

```
package swipNet.utility;

import java.io.*;
import java.net.*;
import swipNet.dcObjects.*;
import swipNet.shipObjects.*;
import swipNet.utility.*;

/**
 * Responsible for sending and receiving of streams on designated network
 * multicast addresses <p>
 * @author LT William G. Wilkins
 * @version 1.0
 */
public class PostOffice extends Object
{

    /**Constructor - Default
     *
     */
    public PostOffice() {}

    /**
     * This method takes Objects like Fire, ShipStatus, etc, a Datagram (or
     * multicast) Packet by using that object toBytes() method;
     * It assumes that a the Socket has already joined a multicast Address
     * @param typeObject the object you want to send, can be fire, flooding etc
     * @param socket the socket that was created within its caller
     * @param address the IP address of caller
     *
     * @return void
     */
    public void sendMulticastPacket(Object typeObject, MulticastSocket socket,
    InetAddress address){

        DatagramPacket packetSend;
        byte buffer[] = null;
        try{

            if (typeObject instanceof Initialize){//0
                Initialize initializeSend = new Initialize();
                initializeSend = (Initialize)typeObject;
                buffer = initializeSend.toBytes();
            }
            if (typeObject instanceof Fire){//1
                Fire fireSend = new Fire();
                fireSend = (Fire)typeObject;
                buffer = fireSend.toBytes();
            }
            if (typeObject instanceof Flooding){//2
                Flooding floodingSend = new Flooding();
                floodingSend = (Flooding)typeObject;
                buffer = floodingSend.toBytes();
            }

            if (typeObject instanceof Hydrapture){//3
                Hydrapture hydraptureSend = new Hydrapture();
                hydraptureSend = (Hydrapture)typeObject;
                buffer = hydraptureSend.toBytes();
            }
        }
    }
}
```

```

    }

    if (typeObject instanceof AirRupture){//4
        AirRupture airRuptureSend = new AirRupture();
        airRuptureSend = (AirRupture)typeObject;
        buffer = airRuptureSend.toBytes();
    }
    if (typeObject instanceof HotRun){//5
        HotRun hotRunSend = new HotRun();
        hotRunSend = (HotRun)typeObject;
        buffer = hotRunSend.toBytes();
    }
    if (typeObject instanceof FastLeak){//6
        FastLeak fastLeakSend = new FastLeak();
        fastLeakSend = (FastLeak)typeObject;
        buffer = fastLeakSend.toBytes();
    }

    if (typeObject instanceof SlowLeak){//7
        SlowLeak slowLeakSend = new SlowLeak();
        slowLeakSend = (SlowLeak)typeObject;
        buffer = slowLeakSend.toBytes();
    }

    if (typeObject instanceof StmRupture){//8
        StmRupture stmRuptureSend = new StmRupture();
        stmRuptureSend = (StmRupture)typeObject;
        buffer = stmRuptureSend.toBytes();
    }
    if (typeObject instanceof RxScram){//9
        RxScram rxScramSend = new RxScram();
        rxScramSend = (RxScram)typeObject;
        buffer = rxScramSend.toBytes();
    }
    if (typeObject instanceof RadSpill){//10
        RadSpill radSpillSend = new RadSpill();
        radSpillSend = (RadSpill)typeObject;
        buffer = radSpillSend.toBytes();
    }
    if (typeObject instanceof ShipStatus){//20
        ShipStatus shipStatusSend = new ShipStatus();
        shipStatusSend = (ShipStatus)typeObject;
        buffer = shipStatusSend.toBytes();
    }
    if (typeObject instanceof EngineeringStatus){//21
        EngineeringStatus engineeringStatusSend = new
EngineeringStatus();
        engineeringStatusSend = (EngineeringStatus)typeObject;
        buffer = engineeringStatusSend.toBytes();
    }
    if (typeObject instanceof CompartmentRigs){//22
        CompartmentRigs compartmentRigsSend = new CompartmentRigs();
        compartmentRigsSend = (CompartmentRigs)typeObject;
        buffer = compartmentRigsSend.toBytes();
    }
    if (typeObject instanceof ShipAtmospheres){//23
        ShipAtmospheres shipAtmospheresSend = new ShipAtmospheres();
        shipAtmospheresSend = (ShipAtmospheres)typeObject;
        buffer = shipAtmospheresSend.toBytes();
    }

    packetSend = new DatagramPacket(buffer, buffer.length, address,
socket.getLocalPort());
    socket.send(packetSend);
}
catch(Exception e){
    System.out.println(e);
}

```



```

        System.out.println("Exception occurred in postOffice -
sendMulticastPacket");
    }
}

/**This method receives an Objects like Fire, ShipStatus, etc and converts it
 * to the appropriate object so the caller can use "instanceof" to determine
 * the correct object
 * @param socket created by caller
 * @param address IP address of caller
 * @return holdObject the object that was received in generic form, use
instanceof to convert
 */
public Object receiveMulticastPacket(MulticastSocket socket, InetAddress
address){

    DatagramPacket packetReceive;
    byte receiveBuffer[] = new byte[1500];
    packetReceive = new DatagramPacket(receiveBuffer,
receiveBuffer.length);
    Object holdObject = new Object();

    try{
        socket.receive(packetReceive);
        /** Must receive the Packet data and call factory to determine what
type of object it is receiving since a cannot directly cast
        * a byte[] array to the appropriate casualty object
        */
        holdObject = PostOffice.factory(packetReceive.getData());
    }
    catch(Exception e){
        System.out.println("Exception occurred in PostOffice -
receiveMulticastPacket");
    }
    return holdObject;
}

/** This method accepts a byte[] array and looks at the first int value to
 * determine what to cast the object to. Each casualty or ship object like
 * Fire, ShipStatus etc. has an instance variable (typeOfDCObject) which is
 * always ordered First. This Method returns an Object that holds the disguised
 * DC casualty or ship object
 *
 * @param ba[] the byte array used as the source for conversion
 * @return holdObject the object that was received in generic form
 */
public static Object factory ( byte ba[]){

    ByteArrayInputStream bis = new ByteArrayInputStream(ba);
    DataInputStream dis = new DataInputStream(bis);
    int determineTheTypeOfObject = -1;
    Object holdObject = new Object();

    try{
        determineTheTypeOfObject = dis.readInt();// looks at first int to see
type of casualty object

        /**
        * Convert to the appropriate Object: Fire, Flooding etc.
        */
        switch (determineTheTypeOfObject){

            case 0:
                holdObject = new Initialize (ba);
                break;

```

```

        case 1:
            holdObject = new Fire (ba);
            break;

        case 2:
            holdObject = new Flooding (ba);
            break;

        case 3:
            holdObject = new HydRupture (ba);
            break;

        case 4:
            holdObject = new AirRupture (ba);
            break;

        case 5:
            holdObject = new HotRun (ba);
            break;

        case 6:
            holdObject = new FastLeak (ba);
            break;

        case 7:
            holdObject = new SlowLeak (ba);
            break;

        case 8:
            holdObject = new StmRupture(ba);
            break;

        case 9:
            holdObject = new RxScram (ba);
            break;

        case 10:
            holdObject = new RadSpill (ba);
            break;

        case 20:
            holdObject = new ShipStatus (ba);
            break;

        case 21:
            holdObject = new EngineeringStatus(ba);
            break;

        case 22:
            holdObject = new CompartmentRigs (ba);
            break;

        case 23:
            holdObject = new ShipAtmospheres (ba);
            break;

        default:
            System.out.println("Invalid Factory Conversion");
            break;
    }
}
catch ( IOException ioe){
    System.out.println("IOException occured in PostOffice - Factory");
}
return holdObject;
}
}

```

Class: Initialize

```
package swipNet.utility;

import java.io.*;
import java.util.*;

/**
 *A class that holds information for Initializing all objects, isolated to its
 * own class to allow for future development with a initializing database and
 * allow reuse of its String array components <p>
 *
 * @author      LT William G. Wilkins
 * @version 1.0
 */
public class Initialize extends Object implements Serializable{

    static int typeOfDCObject = 0;
    static int version = 1;
    static String type = new String ("INITIALIZE");

    String ownerName = new String("default");
    public String multicastAddressOfOwner = new String("default");
    String machineName = new String("default");
    String timeSent = new String ("default");
    String messageToSend = new String("None");

    public String [] dccCasLocArray;
    public String [] dccShipRigForFireArray =
    {"No","Yes - All Spaces"};

    public String [] dccAtmWiLimitsArray =
    {"No limits Met","w/i 90 day Limit","w/i 24 HR Limit","w/i 1 HR Limit"};

    public String [] sceneMicArray;

    public String [] sceneAsstRqdArray =
    {"NONE","5 Personnel to Scene","3 Personnel to Scene","1 Personnel to
Scene"};

    public String [] sceneInjuredPersonelArray;

    public String [] sceneDamageEquipmentArray =
    {"NONE","Electrical SwitchBoard Damage","Pump Damage","Rx Spill"};

    public String [] sceneStatusOfFireArray =
    {"Not Determined","Out Of Control","Spreading to Upper Level","Spreading to
Lower Level",
    "Spreading to Adjacent Compartment","Still Burning","Under
Control","Contained",
    "Fire is Out","No Hotspots","Reflash Watch Stationed w/ CO2","Reflash Watch
Stationed w/ CO2",
    "Reflash Watch Stationed w/ Hose","Reflash Watch Stationed w/ PKP","Reflash
Watch Stationed w/ AFFF"};

    public String [] obaCrewMemberArray;

    public String [] obaTimeRemainingArray =
    {"30 Min","20 Min","15 Min","10 Min","5 Min - EXIT NOW","3 Min","2 Min","1
Min","0 Min"};

    public String [] hoseRRHosesArray;
    public String [] hoseFRHosesArray;
```

```

String [] crewList =
{"NONE", "XO", "ENG", "WEPS", "NAV", "LT MUGGLEWORTH", "LT JONES", "LT DWYER",
"LTJg INDELOCATO", "ETC SMITH", "EMC FOSTER", "MMC ALEMAN", "EM1 Black",
"MM1 Hardy", "MM2 Harris", "MM3 Ericson", "ET1 Bolton", "ST1 Johns", "ST2
Sounds",
"ST3 Blair", "SN Dart", "SN Rine", "SA Pohine", "SR Dair", "SR Tubster"};

String [] shipCompartments =
{"UNKNOWN", "DCC", "DC FWD", "DC AFT", "CSES", "CONTROL", "NAV
CENTER", "FCML", "CREWS MESS",
"TORPEDO ROOM", "AMR", "MANUEVERING", "ERUL", "ERML", "ERF", "TGLO BAY", "COND.
BAY", "MSW BAY", "SHAFT ALLEY"};

public String []
multicastChoices={"228.7.5.4", "228.7.5.5", "228.7.5.6", "228.7.5.7", "228.7.5.8", "
228.7.5.9"};

public String [] rxPowerArray =
{"0%", "5%", "10%", "15%", "20%", "25%", "35%", "45%", "50%", "75%", "100%"};
public String [] pumpLineupArray =
{"2S/2S", "2F/2F", "1S/1S", "2S/0", "2F/0", "1S/0"};
public String [] loopLineupArray =
{"2 Loop", "1 Loop"};
public String [] steamPowerArray =
{"0%", "5%", "10%", "15%", "20%", "25%", "35%", "45%", "50%", "75%", "100%"};
public String [] currentBellArray =
{"A1/3", "A2/3", "AI", "AII", "AIII", "B1/3", "B2/3", "BF", "BE"};
public String [] maxBellArray =
{"A1/3", "A2/3", "AI", "AII", "AIII", "B1/3", "B2/3", "BF", "BE"};
public String [] electricLineupArray =
{"NFPLU", "HPLU", "FPLU w/ MG Sec", "HPBCLU"};
public String [] dischargeRateArray =
{"0 Amps/Hr", "100 Amps/Hr", "200 Amps/Hr", "500 Amps/Hr", "1000 Amps/Hr",
"2000 Amps/Hr", "3000 Amps/Hr", "5000 Amps/Hr"};
public String [] ampsRemainArray =
{"5000", "3000", "2000", "1000", "500", "200", "100", "0"};

public String [] currentOpArray =
{"Normal Patrol", "Coming to PD", "Ship Drills", "Engineering Drills", "Rig
for Deep",
"Rig for Ultra Quiet", "Preparing to Dive", "Preparing to Surface",
"Surface Runs"};

/**Constructor - initializes components that use similiar data, like crews
List
*
*/
public Initialize() {
    dccCasLocArray = shipCompartments;
    sceneMicArray = crewList;
    sceneInjuredPersonelArray = crewList;
    obaCrewMemberArray = crewList;
    hoseRRHosesArray = shipCompartments;
    hoseFRHosesArray = shipCompartments;
}

/** This constructor allows casting to the appropriate object depending on
* its type. It should read in the same order that the toBytes places
* onto the ByteArrayOutputStream.
*/

```

```

public Initialize(byte aBuffer[])
{
    ByteArrayInputStream bis = new ByteArrayInputStream(aBuffer);
    DataInputStream dis = new DataInputStream(bis);

    try
    {
        typeOfDCObject = dis.readInt();
        version = dis.readInt();
        type = dis.readUTF();

        ownerName = dis.readUTF();
        multicastAddressOfOwner = dis.readUTF();
        machineName = dis.readUTF();
        timeSent = dis.readUTF();
        messageToSend = dis.readUTF();
    }
    catch(Exception e)
    {
        System.out.println("Exception occurred in (byte[])");
    }
}

/** toBytes method converts all instance variables within the object to a
 * stream to allow a byte buffer to be sent on the network
 */

public byte[] toBytes()
{
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(bos);

    try
    {
        dos.writeInt(typeOfDCObject);
        dos.writeInt(version);
        dos.writeUTF(type);

        dos.writeUTF(ownerName);
        dos.writeUTF(multicastAddressOfOwner);
        dos.writeUTF(machineName);
        dos.writeUTF(timeSent);
        dos.writeUTF(messageToSend);
    }
    catch(IOException ioe)
    {
        System.out.println("IOException - Unable to convert Object to Bytes");
        return null;
    }

    return bos.toByteArray();
}

public String[] getData(){
    String holdString[] = new String[6];

```

```

        holdString[0] = type;
        holdString[1] = ownerName;
        holdString[2] = multicastAddressOfOwner;
        holdString[3] = machineName;
        holdString[4] = timeSent;
        holdString[5] = messageToSend;

        return holdString;
    }

    public void setOwner (String A, String B){
        ownerName = A;
        multicastAddressOfOwner = B;
    }

    public void setData (String A, String B, String C){
        machineName = A;
        timeSent = B;
        messageToSend = C;
    }

    private void initializeArrays(){

    }

}

```

Class: JDBCBridge

```
package swipNet.utility;

import java.sql.*; //Driver is ids.sql.IDSDriver//Archive is jdk12drv.zip
import ids.sql.*;
import java.util.*;

/**
 *Developed for communications with a database, not implemented yet <p>
 *Some of JDBCBridge code used from [Sayat99]
 * @Modified By      LT William G. Wilkins
 * @version 1.0
 */
public class JDBCBridge {

    Connection      theConnection = null; // the JDBC bridge
    DatabaseMetaData theDBMetaData = null;
    Statement        theStatement  = null;
    ResultSet         theResultSet  = null;
    ResultSetMetaData theMetaData   = null;
    String theStatus;

    public JDBCBridge(String status) {
        theStatus = status;
    }

    public void openConnection() throws SQLException
    {
        try
        {
            Driver drv =
            (Driver)Class.forName("ids.sql.IDSDriver").newInstance();

            String url="jdbc:ids://131.120.27.79:12/conn?dsn='SWIPNetDB'";

            Connection theConnection = drv.connect(url, null);

            //Download the database attributes and create a result set.
            theDBMetaData = theConnection.getMetaData( );
            theStatement = theConnection.createStatement( );
            theResultSet = null;
            theMetaData = null;
            theStatus = "Status: OK";

        }
        catch (SQLException sql){
            handleError(sql);
        }
        catch (Exception e){
            e.printStackTrace( );
        }
    }

    public void closeConnection( ) throws SQLException
    {
        try
        {

```

```

        if (theConnection != null) {theConnection.close( );}
    }
    catch (SQLException sql) { handleError(sql); }
}

public void executeQuery(String sql) throws SQLException
{
    //String sql = "SELECT * FROM SWIPNetDB";

    if (theResultSet != null) {theResultSet.close( );}

    theResultSet = theStatement.executeQuery(sql);

    if (theResultSet != null) {theMetaData = theResultSet.getMetaData(
);}

    String K;
    String W;
    String C;
    String M;
    M = theMetaData.getTableName(1);
    DCCPanelTextArea.append(M + "\n");
    M = theMetaData.getColumnName(2);
    DCCPanelTextArea.append(M + "\n");

    while ( theResultSet.next() ) {

        K = theResultSet.getString("Kids");
        W = theResultSet.getString("Wife");
        C = theResultSet.getString("Cars");

        DCCPanelTextArea.append(K + " " + W + " " + C + " "); //Prints
out a Row

        DCCPanelTextArea.append("\n");
        System.out.println(M + K + W);
    }

    public int executeUpdate(String sql) throws SQLException
    {
        if (theResultSet != null) {theResultSet.close( );}
        theResultSet = null;
        theMetaData = null;
        int result = theStatement.executeUpdate(sql);
        return result;
    }

    public String dumpResult( ) throws SQLException
    {
        String result = "";
        try
        {
            int column_count = theMetaData.getColumnCount( );
            while (theResultSet.next( ))
            {
                boolean first = true;
                for (int i = 1; i <= column_count; i++)
                {
                    if (!first) result += ", ";
                    result += theResultSet.getString(i);
                    first = false;
                }
                result += "\n";
            }
        }
        catch (SQLException sql) { handleError(sql); }
        return result;
    }
}

```



```

    }
    // inserts the fields in order
    String getFieldList(String [ ] fields)
    {
        String result = "(";
        boolean first = true;
        for (int i = 0; i < fields.length; i++)
        {
            if (!first) result += ", ";
            first = false;
            result += fields[i];
        }
        result += ") ";
        return result;
    }

    String getValueList(String [ ] values, boolean [ ] isQuoted)
    {
        String result = "VALUES (";
        boolean first = true;
        for (int i = 0; i < values.length; i++)
        {
            if (!first) result += ", ";
            first = false;
            String value = values[i];
            if (isQuoted[i])
            {
                result += "'";

                // double any embedded single quotes
                int j;
                while ((j = value.indexOf('\')) >= 0)
                {
                    if (j > 0)
                    {
                        result += value.substring(0, j);
                    }
                    result += "'";
                    if (value.length( ) > j + 1)
                    {
                        value = value.substring(j + 1);
                    }
                    else
                    {
                        value = "";
                    }
                }
                result += value + "'";
            }
            else
            {
                result += value;
            }
        }
        result += ") ";
        return result;
    }

    String getNonNullString(int col)throws SQLException
    {
        return nonNull(theResultSet.getString(col));
    }

    String nonNull(String s)
    {
        if (s != null) return s;
        return "";
    }

```

```
    }

    // Handles errors that arise from SQL misrep.
    public void handleError(Throwable t) throws SQLException
    {
        //theStatus.setText("Error: " + t.getMessage( ));
        theStatus = "Error: " + t.getMessage( );
        t.printStackTrace( );
        throw new SQLException(t.getMessage( ));
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[Debus 98] **Feasibility Analysis for a Submarine Wireless Computer Network Using Commercial Off The Shelf Components** by LT Steven M. Debus, MS Thesis, Naval Postgraduate School, Monterey, CA, (September 1998).

[Roemhildt99] **Analysis and Vulnerabilities of Spread Spectrum Wireless Local Area Networks on Surface and Sub-Surface Combatants** by Mark W. Roemhildt, MS Thesis, Naval Postgraduate School, Monterey, CA, (September, 1999).

[Rothenhaus99] **Distributed Software Applications in Java for Portable Processors Operating on a Wireless LAN** by Kurt J. Rothenhaus, MS Thesis, Naval Postgraduate School, Monterey, CA, (September 1999).

[Matthews99] **Analysis of Radio Frequency Components for Shipboard Wireless Networks** by Mark M. Matthews, MS Thesis, Naval Postgraduate School, Monterey, CA, (December 1999).

[Sayat99] **Damage Control and Log Taking Java Applications for Shipboard Wireless LANs** by Hanceri Sayat, MS Thesis, Naval Postgraduate School, Monterey, CA, (December 1999).

[McConnel00] **Testing and Evaluation of Shipboard Wireless Network Components** by Richard J. McConnell, MS Thesis, Naval Postgraduate School, Monterey, CA, (March 2000).

[Kurose01] Kurose, James and Ross, Keith, **Computer Networking: A Top Down Approach Featuring The Internet**, Addison-Wesley Publishing Company, Massachusetts, 2001.

[Minasi00] Minasi, Mark, Anderson, Christa, Smith, Brian, and Toombs, Doug, **Mastering Windows 2000 Server**, Sybex, San Francisco, 2000.

[Bianchi00] Bianchi, Giuseppe, **Performance Analysis of the IEEE 802.11 Distributed Coordination Function**, IEEE Journal on Selected Areas in Communication, vol.18, no. 3, 2000.

[IDS00] IDS Software, **IDS Server Users Guide**, www.idssoftware.com, 2000, pg. 84-88.

[Mangione98] Mangione, Carmine **Performance tests show Java as fast as C++**, <http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf-2.html>, JavaWorld.

[Hughes97] Hughes, Merlin, Shoffner, Michael, and Hamner, Derek, **Java Network Programming, 2nd edition**, Manning, Inc, Greenwich, 1997.

[Mahmoud00] Mahmoud, Qusay H., **Distributed Programming with Java**, Manning, Inc, Greenwich, 2000.

[DCConcepts96] **Damage Control (DC) Concepts For the New Attack Submarine (NSSN)**, American Systems Corporation, Rhode Island, December 1996, Unclassified.

[Morrison00] Morrison, Joline and Morrison, Mike, **A Guide to Oracle 8**, Course Technologies, Cambridge, 2000.

[Sun01] Sun Technologies, **The JDBCTM API Universal Data Access for the Enterprise**, <http://java.sun.com/products/jdbc/datasheet.html>, 2001.

[White99] White, Seth, Fisher, Maydene, Cattell, Rick, Hamilton, Graham and Hapner, Mark, **JDBC API Tutorial and Reference, Second Edition**, Addison-Wesley, 1999.

[Balter99] Balter, Alison, **Mastering Microsoft Access 2000 Development**, SAMS, Indianapolis, 46290, 1999.

[Sadiku95] Sadiku, Matthew and Ilyas, Mohammad, **Simulation of Local Area Networks**, CRC Press, Inc, Boca Raton, 1995.

[Quinn97] Quinn, Liam, and Russell, Richard, **Fast Ethernet**, Wiley Computer Publishing, Inc, New York, 1997.

[Hammond86] Hammond, Joseph and O'Reilly, Peter, **Performance Analysis of Local Computer Networks**, Addison-Wesley Publishing Company, Massachusetts, 1986.

[Tanenbaum96] Tanenbaum, Andrew, **Computer Networks**, Prentice Hall, New Jersey, 1996.

[Guide00] OPNET Technologies, **Wireless LAN (IEEE 802.11) Model Guide**, http://www.opnet.com/products/library/WLAN_Model_Guide.pdf, 2000.

[Stallings97] Stallings, William, **Data and Computer Communications**, Prentice Hall, New Jersey, 1997.

[Jain91] Jain, Raj, **The Art of Computer Systems Performance Analysis**, Techniques for Experimental Design, Measurement, Simulation, and Modeling, Chapter 2, pg. 22-25, John Wiley & Sons, Inc, New York, 1991.

[Peterson00] Peterson, Larry and Davie, Bruce, **Computer Networks: A Systems Approach**, Morgan Kaufman Publishers, California, 2000.

[Products00] Alcatel, Inc, **Enterprise Internetworking Products and Solutions Manual**, 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2. Dudley Knox Library..... Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3. Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5121	1
4. Professor Xiaoping Yun, Code EC/YX Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	3
5. Professor C. Thomas Wu, Code CS/Wq..... Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5121	1
6. Mr. Steve Lose..... Program Executive Officer, Submarines PMS450E Washington Navy Yard 1333 Isaac Hull Ave. S.E. Washington DC 20376	1
7. Mr. Gary Lacombe..... 171 Branch Hill Rd Preston, CT 06365	1
8. LT William G. Wilkins Jr..... 2946 Matthew Lane Lawrenceville, GA 30044	2